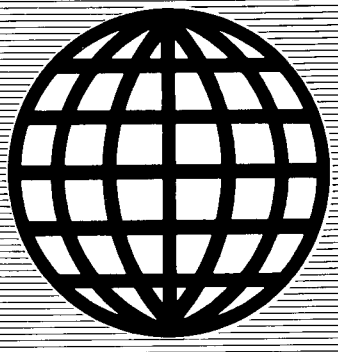


The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 49 March / April 1991 US$3.95

Computer Network Power Protection

Floppy Disk Alignment with the RTXEB and Forth

Motor Control with the F68HC11

Controlling Home Heating and Lighting

Getting Started in Assembly Language

Real Computing

LAN Basics

Z-System Corner

PMATE/ZMATE Macros

Z-Best Software

The Computer Corner

ISSN # 0748-9331

Now $4.95 Stops The Clock On Over 100 GEnie Services

For the first time ever, enjoy unlimited non-prime time* usage of many popular GEnie℠ Service features. For just $4.95 a month. Choose from over 100 valuable services including everything from electronic mail and stock closings to exciting games and bulletin boards. Nobody else gives you so much for so little.

You can also enjoy access to a wide variety of features like software libraries, computer bulletin boards, multi-player games, Newsbytes, and the Computer Assisted Learning Center (CALC) for just $6.00 per non-prime hour for all baud rates including 2400. That's less than half of what some other services charge. Plus with GEnie there's no sign-up fee.

Now GEnie not only gives you the information and fun you're looking for. But the time to enjoy them, too.

Follow these simple steps.

1. Set your modem for half duplex (local echo), at 300, 1200 or 2400 baud.
2. Dial toll free 1-800-638-8369. Upon connection, enter HHH.
3. At the U#=prompt, enter XTX99486,GENIE then press RETURN
4. Have a major credit card or your checking account number ready.

For more information in the U.S. or Canada, call us voice at 1-800-638-9636.

TCJ readers are invited to join us in the CP/M SIG on page 685 and the Forth Interest Group SIG on page 710. Meet the authors and editors of *The Computer Journal!* Enter "M 710" to join the FIG group and "M 685" to join the CP/M and Z-System group.

We'll meet you there!

JUST $4.95

Moneyback Guarantee

Sign up now. If you're not satisfied after using GEnie for one month we'll refund your $4.95.

*Applies only in U.S. Mon.-Fri., 6PM-8AM local time and all day Sat., Sun., and select holidays. Prime time hourly rates $18 up to 2400 baud. Some features subject to surcharge and may not be available outside U.S. Prices and products listed as of Oct.1, 1990 subject to change. Telecommunications surcharges may apply. Guarantee limited to one per customer and applies only to first month of use. GE Information Services, GEnie, 401 N. Washington Street, Rockville, MD 20850. © 1991 General Electric Company.

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Chris McEwen

Technical Consultant
William P. Woodall

Contributing Editors
Bill Kibler
Tim McDonough
Bridger Mitchell
Clem Pepper
Richard Rodman
Jay Sage

The Computer Journal is published six times a year by Socrates Press, P.O. Box 12, S. Plainfield, NJ 07080. (908) 755-6186

Opinions expressed in *The Computer Journal* are those of the respective authors and do not necessarily reflect those of the editorial staff or publisher.

Entire contents copyright © 1991 by *The Computer Journal* and respective authors. All rights reserved. Reproduction in any form prohibited without express written permission of the publisher.

Subscription rates• Within US: $18 one year (6 issues), $32 two years (12 issues). Foreign (surface rate): $24 one year, $44 two years. Foreign (airmail): $38 one year, $72 two years. All funds must be in U.S. dollars drawn on a U.S. bank.

Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 12, S. Plainfield, NJ 07080, telephone (908) 755-6186.

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used trademarks are acknowledged, and we apologize for any we have overlooked.

Apple II, II+, IIc, IIe, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder II, Dos Disk; PlusPerfect Systems. Clipper, Nantucket; Nantucket, Inc. dBase, dBASE II, dBASE III, dBASE III Plus, dBASE IV; Ashton-Tate, Inc. MBASIC, MS-DOS, Windows, Word; MicroSoft. WordStar; MicroPro International. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C, Paradox; Borland International. HD64180; Hitachi America, Ltd. SB180; Micromint, Inc.

Where these and other terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

The COMPUTER JOURNAL

Issue Number 49 March / April 1991

Editorial 2

Computer Network Power Protection 3
Problems, Myths and Solutions
By Wendell H. Laidley.

Floppy Disk Alignment with the RTXEB and Forth 7
Part One
By Frank C. Sergeant.

Motor Control with the F68HC11 13
By Matt Mercaldo.

Controlling Home Heating and Lighting 17
A Personal Embedded Controller System
By Jay Sage.

Getting Started in Assembly Language 19
Making the Jump from High Level Languages
By A. E. Hawley.

Real Computing 21
The PC-532, Minix 1.5 and the 32GX320
By Richard Rodman.

LAN Basics 23
By Wayne Sung.

Z-System Corner 25
Putting the NZCOM Virtual BIOS to Work
By Jay Sage.

PMATE/ZMATE Macros 29
Part Two: Terminology and Utilitiy Subroutines
By Clif Kinne.

Z-Best Software 33
Birth of a New Program
By Bill Tishey.

The Computer Corner 44
By Bill Kibler.

Editorial

By Chris McEwen

Ah, where to begin? It has been a busy two months since the last issue. I have received notes and calls from all over complimenting us on the transition. Thank you all. If I owe you a letter, please bear with me. It will take a while to put everything in its place and calm things down. To add to matters, I was recently informed that I have been assigned to a project team to revamp the MIS program company wide where I work. Nothing like a new job to add to the confusion!

There is one thing we seem to lack here at *TCJ*. There is no place for letters from readers. Since I believe the job of the editor is that of guiding the audiences' attention from one attraction to the next, rather than being an attraction himself, there wouldn't be much purpose in a *Letters-to-the-Editor* column. Instead, we will call it *Reader-to-Reader*. That has a nice ring to it and it fairly represents the involvement subscribers have in our directions here. Feel free to drop us a line. Mark the envelope as *"Attention: Reader-to-Reader."*

Practice What You Preach

When people I know get computers and ask for help, the first thing I teach them is that there is nothing more important than backing up your system on a regular basis. I practice this regularly on my bbs; every Saturday morning is devoted to a ritual of backing up the message system. The lack of a good CP/M backup system keeps me from doing the whole machine every week. Or should I say, what I thought was a lack of a good system. In calling around the country, I found that Roger Warren, sysop of Z-Node 9 in San Diego, not only has devised a scheme but he has gone so far as to modify the BIOS of his Ampro to speed the task by relocating the directory tracks to the center of the floppy disk! Now, that is getting serious. We will be hearing from Roger on this scheme in the near future.

There is a more immediate reason I bring this topic up. I found myself sitting at a dead MS-DOS machine the other week. The beast (I call her Amanda at better moments) had held all the files for this issue just moments before. And then, right at deadline, she decided to blow her file allocation tables. From all we can put together, she took a power surge. No problem for a fellow who preaches regular backups, right? Uh, right. Well, this preacher is out looking for a tape drive. This will happen just once.

The irony of this is that our lead article this issue regards power protection. This is timely, with the spring thunder storms coming along. Wendell Laidley questions whether today's devices might not divert the surge to the delicate data lines through ground. Interesting. Were this to have happened, I would not have lost the FAT, I would have lost the machine! Continuing on, he tells us that MOVs break down in service but fail to give warning. Very nice. The surge protector I have Amanda hooked up to is ten years old. It was the finest I could buy back then, and I certainly didn't see the need to throw it out. Perhaps I should think again.

By the way, I asked one of the major manufacturers of powerline protection devices if they would want to submit something to counter Wendell's position. They declined. Since this is an important topic, I open the door to other perspectives.

Gotta Get a Fix!

Had some correspondence asking where one can go to get CP/M gear fixed. This is becoming more of a problem. The systems we use are getting up in years and there seems to be fewer people who know their way around them. If you are involved in repair of such equipment, let us know. But there are bright spots as well. As you know, Advent stopped selling the TurboROM a while back. That was a crushing blow--there is no single improvement a Kaypro owner can make more important than putting in this ROM. Then, out of the blue, a message was passed around the Z-Nodes. Seems Chuck Stafford has obtained the rights to the ROM and has it back in production. Better yet, the price has dropped to $35. You can call Chuck at his home. The number is (916) 483-0312. Or drop him a line at 4000 Norris Avenue, Sacramento, CA 95821.

We're All Connected

Big news! A couple of special places have been established on GEnie for *TCJ* readers. One, in the Forth SIG (page 710), seems to have been there for a couple of years. It was put up by readers. Art didn't know it was there! I discovered it by accident one afternoon while exploring the system. And the CP/M sysop, Bill Juliani, has established a *TCJ* SIG in his area. This is category 15 on page 685. I am moderating and you are cordially invited. I promise that I will make regular visits to the Forth area as well. Meanwhile, if you are not a GEnie subscriber, you might want to look at their advertisement on the inside front cover. I printed out their index of services in fine print one afternoon and it took some six pages. More to the point, both the Forth Interest Group (FIG) and the most active CP/M group of all the major services live on this system.

Speaking of being all connected, we can look forward to several articles in future issues on Fido-Net and usenet. Mark Burrow of Houston, Texas, promises to talk about a new bbs system for CP/M written in Modula-2 that interfaces quite nicely with Fido-Net. And Andy Meyer of Dunellen, New Jersey was a beta tester for David Goodenough's uucp system. He will tell us of the CP/M usenet tools. With luck, we will see one or both in the next issue.

continued page 40

Computer Network Power Protection

Problems, Myths and Solutions

By Wendell H. Laidley

[Editor's Note: The following paper was presented to the Power Quality conference in October, 1990 and is reprinted here with permission of the author. The points raised in this paper regarding power protection are of importance to everyone using computer equipment. Your computer investment is more than simply money: systems under development may be irreplaceable and lost data could mean a lost business or job. Wendell's position runs counter to prevailing practice. If he is right, the computer using public is at risk. While he specifically addresses networked sites, the weaknesses he cites would could apply equally to any micro computer system with external peripherals. TCJ will offer space for viewpoints from others on this vital topic.]

Introduction

Modern computer networks are uniquely vulnerable to powerline disturbances because they bring together the high energy powerline and sensitive low energy digital integrated circuits. Ordinary surge protection techniques can actually harm computer networks by diverting dangerous surges into delicate network datalines, through the common reference ground. Because of this, computer networks need special attention in powerline surge protection.

Point-of-Use Surge Protection

First generation surge protection strategies followed the conventional wisdom of shunting unwanted surge energy to ground, the ultimate surge sink. The most common practice was to build voltage sensitive shunt components into extension cord power strips, usually using inexpensive MOVs (metal oxide varistors), or sometimes other shunt components such as avalanche diodes (also called TranZorbs), capacitors or even gas tubes.

The components were originally wired only between the line and neutral conductors. Unfortunately, this strategy created a large common mode surge with a risk of arc-over between neutral and ground, so most surge suppresser makers began connecting shunt components between all three powerline conductors, line and neutral and ground, calling this configuration "all three modes of protection".

For simple stand-alone electronic equipment this strategy provided basic protection, particularly if the user understood the limitations of the components, such as the characteristics of MOVs to deteriorate with each surge incident, and of avalanche diodes to fail "open" in the face of a moderate to severe surge, thus exposing the protected load to the full surge. This "three mode" method of surge protection was based on the premise that if voltage differences between the powerline legs at the sensitive electronic load were kept down to acceptable levels, then it would not matter if all three conductors were allowed to "float up" with respect to some absolute or constant reference such as earth ground. In this way the protected equipment would be like a raft without an anchor floating on water--when a wave came along, the whole raft would float up, but so long as there were no anchor to prevent the raft rising, it could float up and down harmlessly. The theory was that if the surge energy or voltage was spread evenly among all three powerline conductors, its effect would be neutralized, since equipment damage should only occur if potential differences arose, and by interconnecting all three lines with shunt components, such differences would be precluded.

The advent of computer networks changed all this by invalidating this fundamental assumption, much as a fixed length anchor line would invalidate the assumption of the free-floating raft in a rising tide. The difference between stand-alone and interconnected computers around from the fact that dataline signals between devices are referenced to ground, and the only "ground" available at the point of use of the sender or receiver is the powerline safety ground conductor, represented by the round pin on conventional 110 volt receptacles.

Problems

Now the practice of shunting surges to the powerline ground becomes unacceptable, because shunting a surge to the powerline ground will cause the ground conductor voltage to rise. This happens because the inductance of the ground conductor between the receptacle and the actual zero impedance ground at the building service entrance will present impedance to the propagation of the high frequency surge, causing the voltage on the ground conductor at the receptacle to rise with respect to true ground at the building service entrance. This is shown in Figure 1, where the voltage gradient along the

Wendell H. Laidley has held positions of systems engineer and project manager through executive vice-president for computer manufacturers and property development firms. He became president of Isomedix, Inc. in 1985 and left in 1986 to found Laidley Interconnect Systems, Inc. Upon the sale of his company in 1989, Wendell joined with Rudy Hartford to found Zero Surge, Inc. to produce powerline protection equipment. He may be contacted at Zero Surge, 103 Claremont Road, Bernardsville NJ 07924.

powerline conductors is shown. In effect, a voltage divider operates, and where the three powerline conductors are of equal length, Figure 1 shows the voltage distribution that will occur.

Since the powerline ground conductor is the only ground available for reference by the signal dataline, diverting surges to it will have the effect of diverting powerline surges directly into the datalines, and thus into the sensitive low voltage internal circuitry of the computer[1].

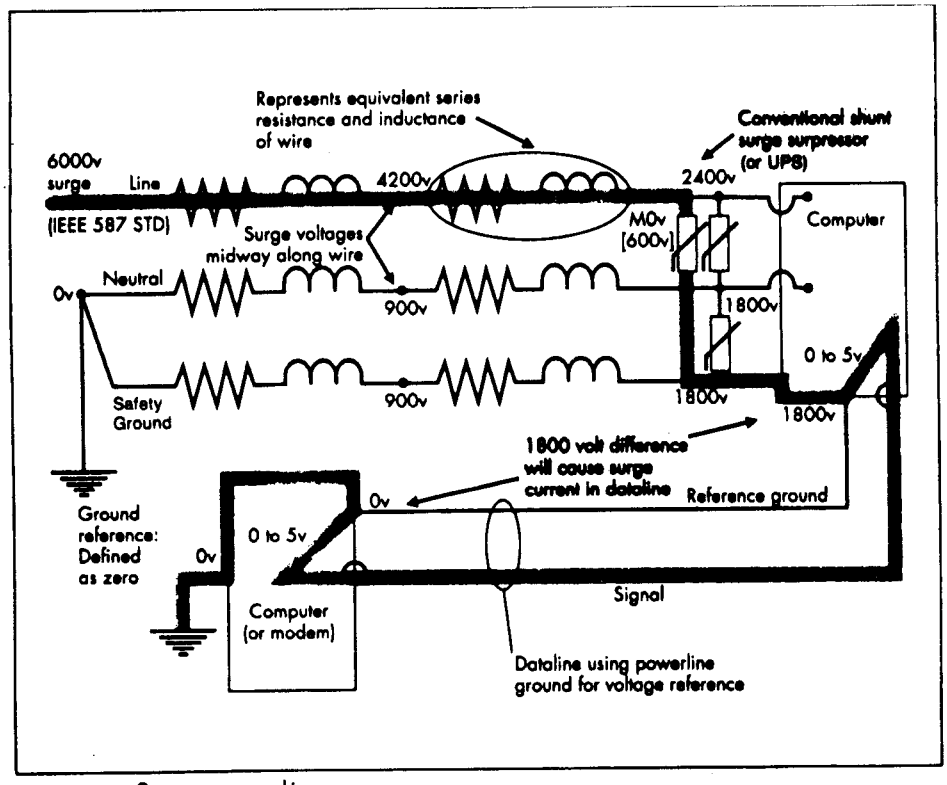


FIGURE 1: Surge coupling

Ground: Surge Sink, or Voltage Reference One, But Not Both

Given this problem, the powerline ground conductor should not be used as a surge sink with interconnected computers where it is needed for voltage reference by datalines. The damage caused by diverting surges to ground in networked electronics was first reported by Francois Martzloff in the 1988 IEEE paper "Coupling Propagation and Side Effects of Surges in an Industrial Building".[2] Martzloff's research team was performing surge tests on an office building over a weekend, and when the office workers returned Monday morning they found their printers did not work and the printer data ports had been damaged. Initially, the research team had not expected that surges they had applied only to the powerline would have damaged datalines. On reflection, they recognized that the powerline surges had indeed been diverted by shunt surge protectors into the printer datalines through the common reference ground.

Problems caused by the interaction of the powerline and datalines through the common ground include physical hardware damage as reported above, and disruption in the form of program lock-ups, data alteration, parity errors and transmission failures. High frequency surges, sometimes with considerable energy, couple into digital circuitry and may alter data in bit streams passing through the CPU. The huge costs of computer network failures were reported in 1989 by Infonetics, a California market research firm.[3] They they reported that local area network downtime costs the average Fortune 500 company $3.48 million annually with the average respondent reporting 23.6 network failures annually, of 4.9 hours average duration. Clearly, these costs deserve attention, and one cause of network failure is powerline surges.

Myths of Surge Protection

Because transient analysis is one of the more complex and less generally understood branches of electrical engineering, the field of surge protection has developed a number of myths over time. Here are some of them, with possible explanations of their origins.

Myth 1: "Any Surge Protection Is Better Than No Surge Protection."

This is perhaps the most reasonable yet the most misleading of all. As explained above, it is shunt surge suppressers themselves which divert powerline surges into datalines, so with networked computers, the wrong design surge suppresser can actually cause computer network failures. With no surge protectors at all, incoming surges will encounter the computer's power supply, which is considerable more surge tolerant than the dataline circuitry. So network users may actually be better off with no surge suppresser than they are with shunt design suppressers which divert surges into datalines and modems.[4]

Myth 2: "UPSes Provide Dependable Surge Protection."

Because a UPS costs far more than a surge protector, it is commonly assumed to provide premium surge protection. Essentially all micro-computer UPSes, 1000vA and under, are a combination of an inexpensive MOV surge suppresser and a battery back-up power source. The MOV surge protection is designed to protect the UPS circuitry, and diverts incoming surges to ground like a common surge protector (see Figure 1). Once on ground, the surge will circumvent the UPS and couple directly into any computer datalines.[5] Since many micro-computer UPSes are used in the context of local area networks, this problem must be solved or the UPS will contaminate the network datalines. Some UPS makers show how surges which encounter the battery of the UPS are effectively eliminated. This is true for the surges which reach the battery, but most are diverted away from the UPS circuitry to ground before they reach the battery. Thus the belief that the battery in a UPS is an effective surge sink is not entirely relevant or dependable. Just like the basic surge suppresser, the UPS protects the computer power supply, but in doing

so, it endangers the datalines.

Another risk exposure with UPSes is the alternative power path around the battery and inverter. So called standby UPSes normally provide direct utility power to the computer, with only the MOVs at the UPS power inlet as surge protection, while on-line UPSes generally have a bypass circuit to enable utility power to flow directly to the load in the event of UPS failure. Both these circuits provide paths to the protected load for incoming surges. In the case of the standby UPS, the path is direct, while for the on-line UPS, the surge must pass through the transfer switch, but these switches are often solid state components with modest tolerance for high energy surges, so they may not prevent a moderate to sever surge from passing through them to the protected load.

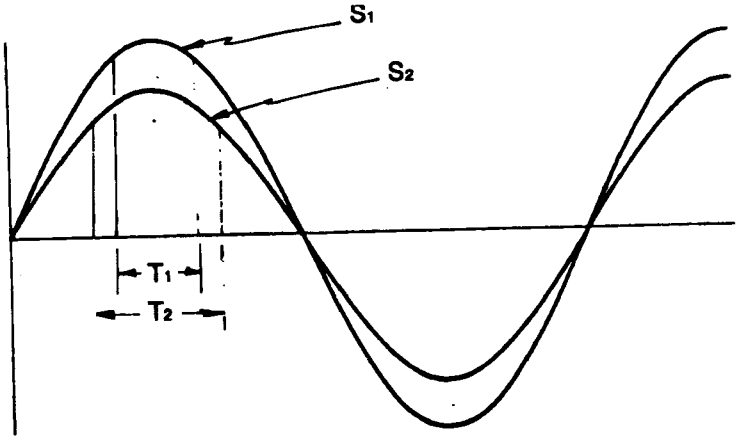
Myth 3: "Transformers Are The Best Surge Protectors."

Transformers are designed to transmit power, not to suppress it. The primary source of surge protection in a transformer is its leakage inductance, which relates to its mass. This inductance provides some surge protection, but less than would inductors specifically designed for surge suppression. A transformer is far from an ideal surge suppresser and presents significant disadvantages such as ringing, regulation, increased source impedance and efficiency loss. They also have substantial parasitic capacitance to ground which can couple surges to ground,[6] and transformers used for surge protection sometimes use MOVs since the transformer may be unable to handle the large voltages in external surges.

The two major advantages of transformers are that they have surge absorbing mass (leakage inductance) and are available as a complete sub-assembly which eliminates the need to design a custom surge processing circuit. The often cited benefit of common mode protection with isolation transformers is somewhat of a red herring issue with computers, as discussed below under Common Mode. Also, while point-of-use isolation transformers re-establish the neutral-ground bond on the transformer secondary, the new ground is rarely connected to a new earth rod, and is almost always connected to the incoming powerline green wire, potentially compromising surge isolation, at least in respect to dataline exposure through the reference ground.

Myth 4: "Voltage Regulating Transformers Are Useful With Computers."

Most modern desktop computers use switch mode power supplies rather than the older style linear power supplies. A switch mode power supply draws from the powerline according to the amount of energy it requires to maintain its output power. In this sense it is a natural integrator of voltage and current, and it compensates naturally and spontaneously for voltage fluctuations. If the voltage drops, it draws current for a longer period, until it replenishes the energy it has put out since its last recharging from the previous cycle of the power wave (see Figure 2). Because of this natural ability of a switch-mode power supply to accommodate varying source voltages, it gains no benefit from a voltage regulating transformer. Instead, the increased impedance inserted into the line by the transformer may hinder the power supply by restricting the current available to the power supply when it calls for current. Switch mode power supplies prefer a low impedance power source that can deliver high current when demanded. Inserting a tap-switching voltage regulating transformer in the powerline will add impedance and restrict the amount of current available to the computer power supply. It may also introduce noise if the tap switch hunts back and forth between adjacent output taps. Computer switch mode power supplies often have a wider tolerance for input voltage than do regulating transformers themselves. Thus the primary benefit of a voltage regulating transformer is its leakage inductance, which is much less than in isolation transformers, but the regulator introduces offsetting disadvantages, and its voltage regulation offers no material benefits.



Area Under Curve S_1 for Time T_1 = Area Under Curve S_2 for Time T_2

FIGURE 2: Switch Mode Power Supply

Myth 5: "Common Mode Surges Cause Computer Problems."

Just as modern switch mode power supplies compensate spontaneously for voltage variations, they also naturally attenuate common mode noise. Desktop computer switching power supplies have five orders of magnitude of common mode noise attenuation built in. The EMI / RFI filter provides two orders of magnitude attenuation, and the high frequency isolation transformer in a switching power supply offers three more orders of magnitude common mode protection at surge frequencies. Common mode sensitivity in computers is a function of slew rate and amplitude of the common mode disturbance. Low voltage, low frequency ground potential differences will not cause disruption or damage, because the primary cause of disruption is coupling, which depends on frequency and amplitude.

Computers are inherently immune to common mode disturbances below a certain threshold, but problems occur when high energy incoming normal mode surges are converted to common mode surges by the action of a shunt surge suppresser. The only source of external surges is normal mode since neutral and safety ground are grounded together at the service entrance (see Figure 1). The high energy of external surges converted to common mode may exceed the common mode tolerance of the computer, but naturally occurring high frequency, low energy common mode noise will not. This problem can be eliminated by keeping shunt surge suppressers off circuits powering computers, thus eliminating the conversion from normal mode to common mode.

Myth 6: "Computer Modem Damage Is Cause By Surges On The Phone Line."

The telephone line is a high impedance circuit which cannot support high energy surges, so they rapidly die away after the inducing source (i.e. lightning) disappears. In contrast, the low impedance powerline provides an ideal propagation network for high energy surges. Also, the telephone line service entrance is protected to under 300 volts, which powerline surges can reach 6,000 volts before they will arc over in 110 fixtures.

The mechanism for most computer modem damage is through high energy powerline surges being diverted to the reference ground and coupling into the digital side of the modem. The elevated voltage then seeks the telephone line ground reference on the analog side of the modem and arcs through the modem.[7]

As a corollary to this, it can be seen that dataline or telephone line protectors may not be effective or may even cause disturbance problems. Dataline protectors which limit voltage between conductors are of limited benefit since externally induced surges, such as from lightning, will appear commonly on both (or all) conductors passing the the inducing magnetic field, and the problem is not between individual conductors, but between the conductors and ground. Similarly, telephone line protectors which provide shunts to the powerline ground, commonly found as cube taps which provide two telephone line jacks and plug into 110 volt receptacles (with plastic rectangular inserts and a conductive ground pin insert to the receptacle) may introduce more disturbance to the telephone line from the powerline ground than they relieve from the telephone line to the powerline ground.

Myth 7: "Signal Ground Is Isolated From Chassis Ground."

Some manufacturers attempt to isolate signal ground from frame ground, but all such isolation configurations have coupling coefficients and dynamic ranges, both of which are likely to be exceeded by high energy external surges. An inspection of most such isolation circuits shows their effectiveness to be generally limited to short duration, low energy noise.

Myth 8: "Transformer Coupled Datalines Are Immune."

These configurations also have coupling coefficients and dynamic ranges, and their effectiveness against low energy noise may not extend to much higher energy surges diverted to the powerline reference ground by ordinary shunt surge suppressers. Even optical dataline converters may be disturbed by such high energy surges.

Myth 9: "The Only Risk From The Powerline Is Hardware Damage."

Computers are vulnerable to data alterations as bit streams pass through microprocessors, and stray power surges can alter data or programs, causing data errors that may never be found or program errors or lock-ups which cannot be traced. The consequential cost of such soft damage can be very high.[3]

Myth 10: "My Surge Protector Is A Permanent Device."

Most point-of-use surge protectors use metal oxide varistors (MOVs) as their primary protection component. This inexpensive (15 cent) component, despite all its strengths, wears out a little with each surge[8] above a very modest threshold that is exceeded many times daily in most environments. As Mark McGranaghan wrote in the premier issue of *Power Quality*, "MOVs cannot handle the energy generated by the switching of utility capacitor banks."[9] Unfortunately, the race among surge protector manufacturers to provide the best protection (lowest let-through voltage) has led them to use lower voltage MOVs which age faster and fail sooner.[10] The normal failure mode for an MOV is thermal runaway to short circuit, and they have been known to cause fire.[11] [Ed.: Consider this as you look at the MOV suppresser sitting on your carpet under your drapes.] MOV based surge suppressers wear out and should be replaced periodically. Unfortunately, the equipment to test an MOV is very expensive (on the order of $20,000) and indicator lights purporting to show that protection is operational are not always reliable, and are sometimes wired across the powerline, indicating only that the powerline is live.

Myth 11: "Nothing Can Stop Lightning."

Of course, this simple statement is true, but in all but the rarest of cases, it may be misleading. Two important qualifiers operate to limit damage from lightning. The first is that a direct lightning strike is extremely rare, and, of course, in that case, equipment and personnel may both be destroyed. But lightning normally manifests itself in the powerline as induced currents caused by the lightning magnetic field on the conductors passing through the field. Thus we normally need only deal with the induced surge, not the lightning strike itself, and the induced surge energy will be limited by the capacity of the conductor to carry the surge energy. The second factor is that surge voltages are limited to 6,000 volts by the arc-over level of fixtures in 110 volt circuits. Thus surge protection need only deal with voltages up to 6,000 volts, and currents determined by the impedance of the source. There are surge protectors available which suppress up to 6,000 volts and unlimited currents to under 250 volts without degradation, without disturbing the critical reference ground.

Myth 12: "You Get What You Pay For."

The assumption that higher priced surge protectors provide greater effectiveness and reliability is not always valid. Almost all surge suppressers under $200 rely on the same fundamental MOV components. Much of the supplementary circuitry is peripheral to the surge protection, such as lights

continued page 39

Floppy Disk Alignment with the RTXEB and Forth

Part One

By Frank C. Sergeant

Introduction

General Description

This paper describes a Floppy Disk Drive Aligner built around the RTX2001A microprocessor. It shows how to add I/O ports and control hardware to generate and measure voltages and time intervals. Examples show how to control the RTX's interrupts and timers and how to put all these parts together to create a smart instrument that provides several improvements over the usual method of aligning drives. A full source code listing and schematic are included. These hardware and software techniques can be applied to almost any RTX project.

A disk exerciser controls which head is active and steps it from track to track. The oscilloscope displays the signal from the drive's read amplifier. These are used with an analog alignment disk (AAD) that contains special patterns needed to align the drive. The Aligner uses a standard AAD and replaces the exerciser and the oscilloscope that are normally used. It could display the results on a self-contained LCD or through a serial line to a PC or terminal.

The RTX2001A provides advantages both during the design stage and in the final product. An RTXEB Evaluation Board with Forth in ROM and a wire-wrap area was used, with an XT clone, as the entire development environment. This could well obsolete expensive development stations. Development was easy because of the interactive access to the hardware and the modular nature of Forth. The RTX is fast, letting software replace hardware (in this application, for example, the ADC and the Peak Detector were done in software).

Alignment

An alignment is done to restore the correct mechanical relationship between the head and the diskette. When a head moves out of alignment the drive may not be able to read disks it wrote some time ago or to exchange disks with other drives. There are two basic head position adjustments: RADIAL (when on track 16, the head should be the correct distance from the center of the disk) and AZIMUTH (the head should be "square" to the track, rather than skewed). At the same time such things as motor speed and the various sensors are checked and adjusted.

The discussion will center around the alignment of a Tandon 5-1/4 inch, 360K drive using the DYMEK DK 502-1 AAD. Generally the same procedure is used for all floppy drives, but a different alignment diskette will be needed for other densities.

Aligning Disk Drives

The Problem

Ordinarily, a drive is aligned using a disk drive exerciser and an oscilloscope. The bulkiness of this equipment makes aligning drives at the user's site awkward, so drives are usually removed and taken to the shop for servicing. This increases the turn-around time and the expense and inconvenience to the user.

One approach to this problem is the digital alignment diskette. This eliminates the need for the oscilloscope and the exerciser. The idea is to run a program on the computer whose drive is being aligned. This eliminates the bulky equipment but makes the technician's work more difficult since the drive must stay attached to its computer. Often there is not proper working room and the drive balances precariously on top of the open computer case. In addition, technicians generally believe that a digital alignment diskette produces inferior results. Often, it is merely used as a quick Go/No Go test to decide whether to take the drive to the shop for an (analog) alignment.

The Solution

The RTX2001A runs fast enough that it can combine the disk drive exerciser functions and the oscilloscope function into a single compact unit. Now, accurate on-site alignment can be done (using the AAD) and this becomes the preferred choice. The RTX chip with some simple support circuitry, on perhaps a 4 x 5 inch board,

Frank Sergeant is a hardware/software consultant specializing in business and/or realtime systems. He is the author/implementor of Pygmy Forth for PC/MS-DOS systems (version 1.3 is available from FIG, GEnie, and fine BBSs and shareware houses everywhere). He has been designing, building, and programming microcomputer systems since the late '70s. One of his greatest joys is replacing hardware with software. He is in the process of porting Pygmy to the Super-8, 68HC11, RTX, etc. His floppy disk drive aligner entry won the RTX design contest. Shortly thereafter he was shocked to hear the RTX was being abandoned by Harris. However, recent conversations with Harris officials have reassured him that it was only future development that was abandoned. Harris has fully, publicly committed to producing the RTX for a minimum of 2-1/2 years. In light of that, Frank breathed a sigh of relief and continues his RTX development work. Frank can be reached as F.SERGEANT on GEnie or through TCJ.

scr # 3601

```
( Adjust the RTX memory map to give the most code space )

17186 DUP H-FENCE !  H !    ( Lower the start of the dictionary)
                            ( to just above the end of the data)
                            ( space needed by the Aligner)

17194 R-TOP !               ( Mark the new top of data space.)
                            ( When we finish loading, THERE )
                            ( will have a value of 17184, which)
                            ( is just under our new value for)
                            ( H-FENCE and H.  R-TOP must be set)
                            ( higher, though, or EBFORTH thinks)
                            ( we've crossed the limit when we)
                            ( really haven't.
```

scr # 3901

The memory map of EBFORTH separates data space from code space (for easy ROMability). During development these are both in RAM, which can be allocated between them by changing a few pointers.

---

scr # 3602

```
( Terminal specific definitions)

: 2EMIT ( hhll -) DUP -8 SHIFT ( xxll xxhh) EMIT EMIT ;

: LOAD ( scr# -)
  DECIMAL    5 ( ie code to request a screen) EMIT    2EMIT ;

: THRU ( 1st last -)
  DECIMAL    6 ( ie code to request a range of screens) EMIT
  SWAP ( last 1st)  2EMIT 2EMIT ;

: AT ( y x -)
  8 ( ie code to request direct cursor positioning) EMIT
  SWAP ( x y) EMIT EMIT ;
: CLS ( -)   9 ( ie code to request clear screen) EMIT ;
```

scr # 3902

2EMIT Break a 16 bit number into 2 bytes and EMIT them.

LOAD Request that host send a screen.

THRU Request that host send a range of screens.

AT Ask host to position cursor at y x co-ordinates. Origin is upper left corner. Height is sent 1st followed by width.

CLS Request that host clear the screen.

---

scr # 3603

```
( Variables )
VARIABLE OUT        ( holds last value written to output port)
VARIABLE TRK        ( holds current disk drive track position)
VARIABLE HEART      ( holds address of active test routine)
VARIABLE Vb         ( holds base voltage for azimuth test )
VARIABLE COMP-FLAG  ( set true if comparator goes high)
VARIABLE #PEAKS     ( EI3-INTW uses it to count wave-form peaks)
VARIABLE OSCALE     ( ''oscilloscope'' scaling factor for raw data)
VARIABLE OTRIG      ( ''oscilloscope'' trigger delay in ms)
VARIABLE WINDOWS
14 ALLOT  ( starting and ending times for azimuth bursts)

 8 OSCALE !  ( reasonable starting value for cat's eye display)
25 OTRIG  !  ( ms from index pulse )
```

scr # 3903

OUT Holds last value written to the output port. This is important because the output port is write-only. We cannot read it back, so every time we write a new value to it, we save a copy in the variable OUT.

TRK Holds current disk drive track. The only time we can read the track position directly from the drive is when it is on track zero (the track0 line goes true). The word RESET steps the head outward, toward track zero until the track0 line goes true and then stores a zero in TRK. Thereafter every time we step the head we also update TRK.

WINDOWS This array holds the starting and ending timer values for each of the four azimuth bursts. This information is discovered by CLOCK and MARK and then used by AZ to measure the azimuth burst amplitudes at the correct times.

---

scr # 3604

```
( Constants )

199  CONSTANT #SAMPLES
     ( # of samples less one to be collected for cat's eye )

( bit-masks for output port devices )
              (   1 CONSTANT  LED    )
              (   2 CONSTANT *DRV0   )
              (   4 CONSTANT *MOTOR  )
                  8 CONSTANT  HEAD
                 16 CONSTANT *DIR
                 32 CONSTANT *STEP
              (  64 CONSTANT *WE     )
              ( 128 CONSTANT *WRITE  )
```

scr # 3904

#SAMPLES Number of measurements to take (less one, to adjust for FOR NEXT) for the cat's eye pattern during one revolution of the disk.

The only output lines we need (connected to the 34 pin disk connector) are HEAD, *DIR, & *STEP. The *WE, *WRITE, & LED bits are connected, but commented out here since we don't use them. The *DRV0 and *MOTOR lines are not connected to the output port. Instead, they are hardwired active (low). If you wanted to put them under program control, they could be connected to bits 0 and 1 (with bit masks of 2 and 4).

These control lines are connected to the lower byte of the output port. The DAC is connected to the upper byte.

replaces essentially all of the hardware of both the disk exerciser and the oscilloscope!

The technician communicates with the Aligner through any terminal or PC, perhaps the PC whose drive he is aligning. Or, the Aligner could show its results on an LCD. By reducing bulk and eliminating complexity and extra cables, the RTX makes on-site alignment more convenient, cutting servicing costs and downtime. Since it is software based, upgrading to handle new types of drives is easy.

How The Exerciser Part Works

The standard 34 conductor cable carries the write-protect, track-zero, index, and read-data signals from the drive to the Aligner and the step, direction, and head select signals from the Aligner to the drive. The Aligner controls the step line to move (SEEK) the head from track to track, and monitors the track-zero switch. With this and the simple terminal interface it can step the drive to the proper tracks to read the AAD's different test patterns and report the status of the drive. To check the motor speed, the time between index pulses is measured. To check the track-zero switch, the track-zero line is monitored while the head is stepped toward track zero.

How The Oscilloscope Part Works

The Aligner measures the amplitude of the read signal by doing analog to digital conversion in software, using the DAC and comparator. These readings are then displayed as a histogram using columns of 'X's on a character based terminal, or as a higher resolution picture on a graphics terminal. This provides the cat's eye and azimuth burst displays the technician needs.

Interfacing to the RTX

I/O Port Hardware

Although the ASIC bus has some interesting possibilities for on-chip integration, for our purposes just think of it as the I/O bus. This is also called the G-bus. It is made up of 16 data lines, 3 address lines, and 2 control signals. The 3 address lines mean that you can have a maximum of 8 I/O addresses. Each I/O address can access 16 inputs and 16 outputs, for a total of 32 bits per address.

Since the Aligner only needs 4 inputs and 14 outputs (and we could get by with fewer) we don't need to decode the G-bus address lines at all, only the control signals are needed. GIO* goes true (low) on either a read or write of a G-bus address. GR/W* goes low for a write and stays high otherwise. We latch data to the output port only when both GIO* and GR/W* are low. We enable data from the input port only when GIO* is low and GR/W* is high. Because of this approach any external G-bus address will work with our ports. Throughout the listing G-bus address 31 is used, but any of the addresses 24 through 31 will work equally well. Be careful not to use addresses 0 through 23 as they address internal devices on the ASIC bus (that is, they address processor registers). 74HC373 latches are used for all the ports. Two of them face outward, providing 16 outputs. One faces inward, providing 8 inputs. Note, with this method the outputs are write-only. If you try to read them, you will get the inputs at the same address.

I/O Port Software

31 G@ reads the input port and leaves it on the stack. u 31 G! stores the number u to the output ports. Only 8 bits of the 16 that are available are connected for the input port. All 16 bits are connected for the output port. Usually we will not just store a number to the output port; we will also save a copy in the variable OUT. This keeps track of what we last wrote, allowing us to change selected bits without disturbing the others.

Here is an example of how to use the I/O ports. Look at the schematic and note that the least significant bit (LSBit) of the output port is connected through an inverter to an LED. (The base is hexadecimal for these examples.) The simplest way to flash the LED is to type

```
HEX
31 G!       ( turn on LED)
31 G!       ( turn off LED)
31 G!       ( turn on LED)
31 G!       ( turn off LED)
```

over and over. If the LED goes on and off, the output port address decoding (and our understanding of it) is verified. The above approach has the flaw that we might be altering other bits (that affect devices other than the LED). We can't just do

```
31 G@       1  OR    31 G!
 1 G@    FFFE AND    31 G!
```

because we would not be reading the output port. We would be reading the input port. So, we save a copy every time we store something to the output port, and we read this copy when we want to know what we last wrote. This would work:

```
VARIABLE OUT            ( define a variable to keep the copy)
FFFF DUP 31 G!   OUT ! ( put bits in a known state )

OUT @     1  OR   DUP 31 G!  OUT !   ( LED on)
OUT @  FFFE AND   DUP 31 G!  OUT !   ( LED off)
OUT @     1  OR   DUP 31 G!  OUT !   ( LED on)
OUT @  FFFE AND   DUP 31 G!  OUT !   ( LED off)
```

Naturally, using Forth, we won't keep doing all this typing. Instead we will hide these details in a convenient definition, as follows:

```
: ON  ( bit-mask -)       OUT @  OR   DUP OUT !   31 G!  ;
: OFF ( bit-mask -)   NOT OUT @ AND   DUP OUT !   31 G!  ;
```

Since the LED is connected to the LSBit of the port, the bit-mask for it is simply 1. Note that OFF takes care of inverting it for the AND so that we don't have to think in complements. Now we can flash the LED with

```
1 ON
1 OFF
1 ON
1 OFF
```

but, why should we have to remember that the bit-mask for the LED is 1 ("because it is easy to type" you might reply)? We'll hide that detail too:

```
1 CONSTANT LED
LED ON
LED OFF
LED ON
```

```
scr # 3605
( read disk status from input port )

: P@ ( - u) 31 G@ ; MACRO  ( ''P fetch''  read from input port)

: INX?  ( - f) P@ 1 ( ie bit mask for index line) AND 0= ;

: ?NOT-INX ( -) BEGIN     INX? 0= UNTIL ;

: ?INX     ( -) BEGIN        INX? UNTIL ;

: SYNC ( -) ?NOT-INX ?INX ; ( wait for start of index pulse)

: TRK0? ( - f) P@ 2 ( ie bit mask for trk 0) AND 0=  ;

: WP?   ( - f) P@ 4 ( ie bit mask for write protect) AND 0=  ;
```

scr # 3905

Read Disk Status - The input and output ports are separate ports accessed by a read or write of ASIC address 31. The GR/W line determines whether the 16 input bits or the 16 output bits will be accessed. As long as we don't need more than 16 'I's and 16 'O's, we can have 32 I/O lines without decoding the ASIC address lines.

To read a status line, we read all 16 bits and then isolate the one we are interested in by ANDing with its bit-mask. Since these are active low, we follow with 0= so we will have a true flag only if the line we are interested in is low.

```
scr # 3606
( write to output port )

: P! ( u -) 31 G! ; MACRO ( ''P store''  write to output port)


( set or clear a specific bit without disturbing the others)

: ON  ( bit-mask -)      OUT @ OR  DUP OUT ! P! ;      -1 ON
: OFF ( bit-mask -)  NOT OUT @ AND DUP OUT ! P! ;
```

scr # 3906

P! Write the value on the stack to the output port.

ON This reads the variable OUT to find out what we wrote to the output port last time, ORs in the new bits we want to turn on, saves the result in OUT for use next time, and writes the result to the output port. As soon as ON is defined, we use it to write all ones to the output port so it will be in a known state. The point is that ON let's us turn specific bits on without disturbing the state of the other bits.

OFF Does the same thing except it turns specific bits off without disturbing the others.

```
scr # 3607
( select active disk head - either head zero or one )

: H0 ( -) HEAD OFF ;

: H1 ( -) HEAD ON ;

: -H ( -) ( ''toggle head'')
   OUT @ HEAD AND IF H0 EXIT THEN H1 ;
```

scr # 3907

H0 Make head zero the active head.

H1 Make head one the active head.

-H Switch from one head to the other.

```
scr # 3608
( step head to selected track )
: STEP ( -)  *STEP OFF 5 MS *STEP ON 10 MS ;
: +STEP ( -) *DIR OFF STEP 1 TRK +! ;  ( move head inward  )
: -STEP ( -) *DIR ON STEP -1 TRK +! ;  ( move head outward )

: SEEK ( track -)
   DUP  TRK @ - ( trk  #steps)
   *DIR OVER  0< IF ON ELSE OFF THEN ( trk  #steps)
   ?DUP IF     ABS 1- FOR ( trk) STEP NEXT THEN    TRK ! ;

: RESET ( -) BEGIN   TRK0? 0= WHILE   -STEP REPEAT   0 TRK ! ;

: TRK1  ( -)  1 SEEK ;
: TRK16 ( -) 16 SEEK ;
: TRK34 ( -) 34 SEEK ;
```

scr # 3908

Whenever the *STEP line is pulsed low, the heads move one track either in or out. The *DIR line determines the direction. When *DIR is low, when *STEP is pulsed, the head will be stepped outward (toward lower numbers). Otherwise, when *STEP is pulsed, the head will move inward (toward higher numbers).

SEEK Move the head to the specified track. It compares where it is (based on the current value of TRK) to where you want it to go, to determine which direction and how many STEPs. All these words that affect the track location must update TRK. Before stepping the heads the 1st time, RESET should be performed to synchronize the head location with TRK.

```
scr # 3609
( Clamp value between allowable limits )

: CLAMP ( n  lower-limit  upper-limit  -  n')
  >R MAX R> MIN  ;
```

scr # 3909

CLAMP Force the number to be in bounds.

```
scr # 3610
( accept a number from keyboard)

: #IN ( - u)
  PAD 1+ 5 EXPECT    0 0 PAD CONVERT 2DROP ;
```

scr # 3910

#IN Accept up to 5 keystrokes and convert to a number. This word allows the menu to prompt the technician for a number. This is used in the words OTRIG and OSCALE to get the new values for the cat's eye trigger delay and the scaling factor (Range) for the cat's eye graph.

```
LED OFF
```

Now, if we connect another device to the next bit over, it would have a bit-mask of 2.

```
2 CONSTANT NEW-DEVICE

LED ON
NEW-DEVICE OFF
```

We can turn one device on or off without disturbing the other. Look at screen #3636. This carries this same example a little further, showing how to read the drive's *Index line and make the LED go on and off in sync with it. With a disk spinning in the drive the LED will flash 5 times a second.

TIMERS

The RTX provides 3 on-board timers. They run all the time, counting down to zero and automatically reloading with the last value stored in them. They can count external events, but in this application they just count RTX clock cycles. Timers are 16 bits wide. Each can be the source of an interrupt or can be read directly. Here's how to set timer 1 to count down with a one millisecond time period

```
8000 TC1!
```

This assumes an 8 MHz clock, which gives 8 RTX cycles per microsecond and 8000 per millisecond. You only have to initialize the counter once, then each time it counts down to zero it will automatically reload with the same value. You can read the counter with

```
TC1@
```

which leaves the value on the stack. If you want to find exactly how long a group of instructions takes to execute you can initialize the counter before the group and then read it afterwards, as follows

```
: TIMEIT ( - cycles)
  0 TC1!   <group of instructions to be measured>
  TC1@ NEGATE  ;
```

The NEGATE adjusts the down-counting value to the number of elapsed cycles. Timer 1 has priority level 8, so its interrupt routine would be installed, enabled, and disabled as follows

```
  '  T1-INT   8 !INTERRUPT      ( install from the keyboard)
[']  T1-INT   8 !INTERRUPT      ( install from a definition)

TIMER1 UNMASK     ( enable)
TIMER1 MASK       ( disable)
```

The timer interrupts are edge triggered. This means that if the timer has been running awhile before you enable its interrupt then an interrupt will be pending. It will wait and wait until you finally enable it, and immediately jump to the handler. So you get an 'extra' interrupt.

Interrupts

There are lots of interrupts. Five of them are connected to external pins EI1 through EI5. EI5 is in use on the RTXEB for serial communication with the host. The Aligner only uses EI3. These are level sensitive, active high.

Interrupt handling in Forth on the RTX is very easy. On a regular micro you treat interrupt handlers cautiously. They have to be installed just right. Then you must figure a way to trigger them to see if the handlers work. You can't just use JSR or CALL as the handlers must end in return from interrupt instructions rather than return from subroutine instructions.

However, on the RTX, the regular return instruction is the same for both interrupt handlers and subroutines! This allows you to call the handler directly--even from the keyboard--rather than setting up complex test scaffolding. For example, here is an interrupt routine to set a variable and then disable itself

```
: EI3-INT ( -)    -1 COMP-FLAG !   EI3 MASK   ;
```

It can be tested from the keyboard by setting COMP-FLAG to zero then typing EI3-INT. Then reading COMP-FLAG, again from the keyboard, to see if it really has a -1 in it. In this application, since the store (!) operator does not affect the carry flag, nothing needs to be saved and restored. This external interrupt (EI3) has a priority level of 10, so the above handler can be installed with

```
  '  EI3-INT 10 !INTERRUPT     ( from the keyboard)
[']  EI3-INT 10 !INTERRUPT     ( within another definition)
```

It takes the address of the handler and stores it in the level 10 slot. EI3 UNMASK enables the routine and EI3 MASK disables it. This can be done from the keyboard or from within another routine or program. Friendly? Yes. Difficult? No. Fast? Ah, that's where the RTX really shines. The latency is 1 to 2 cycles (125 to 250 ns with an 8 MHz clock) max and the return is usually free.

The Aligner--Hardware Components

A standard 34 conductor ribbon cable connects the Aligner to the drive. This connects the status signals to the RTX input port and connects the control signals to the RTX output port. In addition, a test clip is used to connect the drive's head read amplifier signal to the Aligner.

Disk Drive Status Signals

Four inputs are used to read the drive status signals. These are *Index, *Track0, *Write-Protect, and *Read-Data. These are active low. *Index pulses low whenever the index hole in the diskette is lined up with the hole in the jacket (an LED shines through the hole, activating a sensor). This is the main synchronization signal. *Track0 is low whenever the head is moved outward enough to activate the track zero switch on the drive (which happens, ideally, whenever the head is sitting on track zero). This sensor is a micro switch. *Write-Protect is low whenever a diskette in the drive has its write protect notch covered. This sensor can either be an LED or a micro switch. *Read-Data is the raw read signal that pulses low briefly whenever the drive detects a flux transition on the diskette. These are all open-collector digital signals. The Aligner has pull-up resistors on its input port so that these signals can be read correctly.

to be continued next issue.

Disk Drive Control Signals

Five outputs are used to send control signals to the drive: Direction, Step, Write Data, Write Enable, and Side Select. These are all digital signals. The *Motor On and the drive select control signals could be connected to outputs, but are not; they are hardwired active.

Signal Conditioner

Only one analog signal is needed from the drive: from the head read amplifier. This is picked up with a test lead or alligator clip (from TP1 or TP2 on Tandon drives). After conditioning and comparing, this will trigger an interrupt on EI3.

The signal we need to condition comes from the output of

```
scr #  3611
( DAC!  write to 8 bit R-2R DAC )

: DAC! ( u -)
  7 FOR 2* NEXT ( u*256)
  OUT @ 255 AND     OR     P! ;
  ( put it in high half of output port; don't change low half)

(    note that we do not save the dac value in OUT as we do   )
( for the disk drive control bits - the lower byte remains    )
( undisturbed and the lower byte of OUT is still valid.       )
```

```
scr #  3911
Digital to Analog Conversion

This system uses an 8 bit DAC built with an R-2R
voltage divider network, using 47K ohm SIP resistor
packs. 2R equals 47K and R equals half of that (2 47K
resistors in parallel). This gives a 256-step output
between GND and the power supply. This is further
scaled by a fixed voltage divider (100K & 10K) to cut
the output by about an eleventh (a potentiometer could
be substituted), giving a closer match to the signal to
be measured.

A similar result can be achieved by using an integrated
8-bit DAC.

DAC!    Set the output voltage level.
```

```
scr #  3612
( EI3-INTW          interrupt handler used by CLOCK )
            ( save time when each peak starts, starting at PAD)
      ( the beginning of each peak will trigger this interrupt)

: EI3-INTW ( -)
  TC1@  CR@   ( time cc)
  1 #PEAKS +!
  IMR@ ( time cc imr)
  EI3 NOT IMR!              ( ie mask all interrupts except ei3)
  ROT PAD #PEAKS @ 2* + ( cc imr time a) !    ( cc imr)
  BEGIN    CR@ 0>              (    ie msbit of CR will be set)
                               ( as long as comparator is high)
     TC1@ 1000 U<  ( ie bail out if timer counts down too far)
  OR UNTIL                    ( loop until wave peak goes away)
  IMR! CR!   ;
```

```
scr #  3912
EI3  interrupt routine to map windows for azimuth
bursts

EI3-INTW    (''W'' for ''window'')  This stores the
current timer value at PAD and waits for the comparator
to go low again, so we don't count a wave more than
once. These timer values represet the count-down values
from the start of the index pulse. All of the peaks of
interest occur in about the 1st two milli-seconds.

This int handler just stores the values. It is set up
by CLOCK. Then MARK is used to figure out from these
times where the azimuth bursts occur.
```

```
scr #  3613
( CLOCK  find when azimuth bursts occur )
: CLOCK  ( - )
  Vb @ 4 + DAC!     ( set dac so any wave peak will trigger int)
  PAD 200 ERASE     ( clear at least the 1st few bytes for data)
  ['] EI3-INTW  10 !INTERRUPT       ( install interrupt handler)
  0 #PEAKS !                          ( initialize wave counter)
  SYNC                                   ( wait for index pulse)
     0 TC1! EI3 UNMASK       ( start timer and allow interrupts)
  2 MS                      ( 2 milli-seconds is plenty of time)
  EI3 MASK   ;                          ( stop interrupt handler)
```

```
scr #  3913
Find when azimuth bursts occur

CLOCK      Set DAC to a value low enough that every wave
peak should trigger the comparator. Set up the EI3
interrupt handler. Wait for the index pulse (SYNC).
Initialize timer1. Note that timer1 is not enabled as
an interrupt source, but that the EI3 interrupt
(connected to the comparator's output) reads timer1 to
find out when the current wave peak occurs. We let it
run for ''2 MS'' (actually longer because of the time
spent in the interrupt handler) and then close down the
operation by masking the interrupt.

The data we have gathered will be processed by MARK.
```

```
scr #  3614
( EI3-INT  note it if comparator ever goes active )
: EI3-INT ( -)
  -1 COMP-FLAG !      EI3 MASK   ;

( this is just about the ideal size for an interrupt handler)
```

```
scr #  3914
Note whether the comparator ever goes active.
EI3-INT     This is a very simple interrupt handler. If
it is invoked (by the comparator going active) it notes
that fact by setting COMP-FLAG true. Then, it disables
itself. It is used to let us know if the disk drive
signal exceeded the voltage reference level set by the
DAC at any time during a particular time interval. It
turns itself off so it won't be invoked repeatedly
during the high part of a wave.

On the RTX, unlike most processors, the return from
interrupt instruction is the same as a return from
subroutine. This allows an interrupt handler to be
tested from the keyboard just like any other Forth
word!
```

Motor Control with the F68HC11

Second of a Series

By Matt Mercaldo

As I was going through the development cycle for this article on driving a stepper motor, I realized how apparently complex it is to work with a single board computer. Fortunately the NMI series of singleboard computer is predictable, and when approached logically, works quite well with most PCs. This article will serve as an introduction to working with the line of NewMicros F68HC11 based single board computers. The typical development environment, the assembler and how Forth does assembly language will be reviewed. Finally a stepper motor will be spun as promised in the first article of this series.

Max-Forth, a PC and You

The single board environment has three components: source code, a communications program, and Max-Forth on the single board computer. All of my development is done from the Mac with MicroPhoneII by Software Ventures, but the principles and settings apply to most platforms and most communications packages. I chose MicroPhoneII because of its powerful scripting language that eases the "pain" found in the development cycle of having to download modules, being attentive to errors and the like.

The MaxForth Side

Max-Forth is a Forth interpreter. It accepts characters from the serial line at 9600 baud (8 bits, no parity). Characters combine to form commands. These commands or Words (Words are another name for Forth commands) are delineated by spaces. A line is interpreted upon reception of a carriage return character. Word names can be composed with any group of characters other than a space character or a carriage return character. Max-Forth echoes all characters that it receives. When Max-Forth receives a carriage return and interprets the line, it may return a message--it will always send a line feed character (^J) last.

Assume the word HEX is typed at the keyboard of your communications program when connected to Max-Forth on a NewMicros board. The following sequence ensues: (^M is a carriage return, ^J is a line feed, _ is the cursor.)

```
Character typed                    Max-Forth's echo
H_                                      H
HE_                                     E
HEX_                                    X
HEX<cr>                                 ^M^J OK ^M^J
HEX  OK
```

The PC Side

This information on how Max-Forth behaves is very important when deciding on a communications program. If the communications program could wait for a sent character to be echoed, and upon reception of a linefeed (^J character) send a new line, a file could be automatically download to Max-Forth for interpretation. Most communications programs do just that. They can be configured to wait for echoed characters and send the next line after receiving a ^J character. In MicroPhone II, a carriage return must be sent after the line is sent. (I don't really know why this is, but it works!)

The Source File

Different text editors maintain text information differently. In the typical Mac Edit program, a line is ended with a carriage return character only. In other text editors, a line feed may be added to form a carriage return, linefeed pair. Some text editors put a tab character in the text upon reception of a tab character, others put the default number of spaces into the text upon reception of a tab character. Two things that Max-Forth will not accept are linefeed characters and tab characters. Assume that Max-Forth receives <tab>HEX<cr> as a new line to be interpreted from a file. Max-Forth will try to find <tab>HEX in its dictionary of Words - the goal is to find and execute the word HEX. Max-Forth will not find <tab>HEX in its dictionary--it will find HEX. The same applies to linefeed. (The tab caused much anguish when I started to use MicroPhoneII. The communications package I was using before converted the tab characters in my source code to spaces, MicroPhoneII didn't. Nothing would interpret! I thank the programmer who thought up Find and Replace! <control> <tab> sets the find portion to locate Tab characters in the find and replace dialog of Edit.)

In Review of Max-Forth, a PC and You

In review, a source file, a communications program running on a PC, and Max-Forth running on the F68HC11 single board computer from NewMi-

Matthew Mercaldo is employed by a huge firm. With a small group, he develops software tools for field service engineers to do their thing. At 4:30 or 5:00 p.m., when the whistle blows, his thoughts race toward the edge. He dreams of articulated six legged walking beasts, electronic brains that can fend for themselves, and the stuff of "U.S. Robots and Mechanical Men." Someday he dreams of running power out to his garage, and with his wife and a select group of friends, opening his own automoton shop - and thus partially fulfilling his childhood dreams. (Plutonium, Tritium and the like are still not available for public "consumption"; but seeing the moons of Jupiter would be spectacular in one's own starcruiser!)

cros are the components required to develope software on the F68HC11. If the source file has tabs and linefeeds or any hidden character in it, they must be gotten rid of in the editor, or accounted for in the communications program by elimination or replacement. Max-Forth does not want to see these characters in its incoming interpreter stream of characters. The communications program should be configured to wait for the echoed character before sending the next character, and to send a new line upon the communications program's reception of a line feed character (^J).

Forth Assembler, Max-Forth and You

In a previous article *TCJ* readers were presented with my bias on assembler, Forth and Forth assembler. With the information in that article as a base, the necessary assembler basics for this project will be reviewed. The basics will cover Max-Forth's register usage and Data Stack manipulation.

The Stack and its Usage

Max-Forth uses a sixteen bit (two byte) data stack. This means that data objects must be added to or taken from the stack in sixteen bit (two byte) increments. The desired data object on the stack may only be one byte, but upon removal, that removal of the data object must be in a two byte increment. Max-Forth reserves the use of register 'Y' as its data stack pointer. If the Y register must be used, make certain that it is saved going into the assembler routine, and restored upon exit of the assembler routine. This can be done by the PSHY and PULY combination of assembler words.

The 68HC11 puts the least significant byte in the high memory location of a two byte word. In order to access a character on the stack use the following as a guide line:

```
0 ,Y LDD
```

The D register is actually the A and B registers combined. The A register is the most significant byte and the B register is the least significant byte. With the above operation B will contain a character if one is to be found on the stack. When removing an element from the stack, copy the element into a convenient register and increment the Y register twice.

```
0 ,Y LDD        ( get the stack element into D ;)
                ( A is msb - B is lsb  )
INY INY         ( update the stack pointer )
```

When adding an element to the stack, decrement the stack pointer twice before adding the element:.

```
DEY  DEY        ( update the stack pointer for adding)
                ( a new element )
0 ,Y STD        ( put a new element from D onto the stack )
```

This use of the stack can be found in the code found in listing 2.

Approaches to Inline Assembler

There are two types of assembler words in Max-Forth: Those that Max-Forth can run from the interpreter, and those that Max-Forth cannot run from the interpreter. The words that Max-Forth cannot run from the interpreter are used for interrupts and routines supporting interrupt handlers. These interrupt routines are comprised of code put into the Forth dictionary following a CREATE or CODE Forth dictionary header definition. For an excellent treatise on the Forth dic-

Listing 1.

```
COLD

FORGET TASK
HEX
100 1C !
50 1E !
280 DP !

    DECIMAL
    20000 CONSTANT /10MS     ( 2000 ticks of free-running counter     )
    HEX
    ' /10MS @ 026 !          ( Point address 26 at eclock ticks per   )
                             ( 10 ms. Contents of 26 was supposed to  )
                             ( time writes to EEPROM but version 3.3  )
                             ( F68HC11 has bug--it uses 26 as a ptr.  )

: EE!   ( n addr -- )
    2DUP @ =
    IF    2DROP    ( was already equal to n )
    ELSE
        OVER 100 / OVER EEC!    1+ EEC!
    THEN ;

( _____Setting an Interrupt Vector_____ )

( Define each interrupt handler using CREATE, so that the name      )
( of the interrupt handler leaves the address of the code.          )
( Install the handler into the interrupt vector table by            )
(           <code address> <vector name> VECTOR                     )
( For example, to create pulse interrupt handler:                   )
(           CREATE PULSAR                                           )
(               ASSEMBLER                                           )
(               <code>                                              )
(               RTI                                                 )

( Assume that the interrupt vector for PULSE is at address FEEE in the )
( 68HC11.  To get the word >PULSE:                                      )
(           FFFE @ CONSTANT >PULSE                                      )
( Max-Forth points the interrupt vectors into the 11's EEPROM           )

( To install the PULSE interrupt handler:                               )
(           : INSTALL-PULSAR                                            )
(               PULSAR >PULSE VECTOR ;                                  )
( The system reset code should then execute INSTALL-PULSAR.             )

  : IV   ( primary-address <name> -- ; define an interrupt vector )
    <BUILDS    ,    DOES>    ( -- secondary-address )    @ @ ;

FFE8 IV >TOC1

: VECTOR    ( code-addr secondary-vector -- ;       )
    7E OVER EEC!          ( put jump opcode         )
    1+ EE! ;              ( put address             )

( Registers in the F68HC11's register map )
B00E CONSTANT TCNT     ( Free running timer - 16 bit              )
B023 CONSTANT TFLG1    ( Timer Interrupt register - 8 bit         )
B022 CONSTANT TMSK1    ( Timer Interrupt Enable register - 8 bit  )
B016 CONSTANT TOC1     ( Timer output compare register - 16 bit   )
0080 CONSTANT OC1F     ( Interrupt enable flag bit for Timer one  )
```

Listing 2.

```
HEX

( Delay of interrupt ticks before next step state is initiated )
VARIABLE DELAY_TIME   10 DELAY_TIME !
( Offset to be added to TCNT -- free running timer -- for next )
( Timer Output Compare interrupt.                              )
VARIABLE TIMER_OFFSET   100 TIMER_OFFSET !

( The Motor Control Block structure. )
000 CONSTANT NEXT_MOTOR    ( Pointer to Next MCB )
002 CONSTANT ACTION        ( Pointer to an action to carry out )
004 CONSTANT SENSOR        ( Pointer to a sensor action )
006 CONSTANT >>PORT        ( Pointer to the NMIS 7040 Port )
008 CONSTANT TIMER         ( Pointer to a timer variable )
00A CONSTANT STEP_BIT      ( The active motor bit of the NMIS 7040 )
00B CONSTANT MCB_SIZE      ( Size of this data structure )

8000 CONSTANT >PORT        ( Pointer to the NMIS 7040 Port )

CREATE ANCHOR_MCB MCB_SIZE ALLOT  ( The Base MCB for a multiple device    )
                                  ( System.  Each Device has its own MCB. )
                                  ( Each MCB is linked to another.  At a  )
                                  ( Regular Heart beat interrupt, this    )
                                  ( MCB is used to Reference the rest of  )
                                  ( the list of MCBs.  The list is        )
                                  ( circular so the anchor MCB's action   )
                                  ( will also be fired last.  The anchor  )
                                  ( MCB's action is an RTI.               )
                                  ( All MCB's actions must fire before    )
                                  ( the next interrupt fires. )

VARIABLE CURRENT_MCB       ( This is the currently active MCB during the )
                           ( MCB interrupt cycle. )

CREATE FIRST_MCB MCB_SIZE ALLOT   ( The MCB we will use for the NMIS 7040 )
                                  ( Stepper motor example. )

VARIABLE >_STEP          ( Forward References )
VARIABLE >_WAIT
VARIABLE >LAST_ACTION
VARIABLE >SENSOR_ACTION

( Enable Interrupts; This word enables all Interrupts )
CODE EI  ( -- )
ASSEMBLER
   CLI
   NEXT ^ JMP
END-CODE

( Disable Interrupts; This word disables all Interrupts )
CODE DI  ( -- )
ASSEMBLER
   SEI
   NEXT ^ JMP
END-CODE

( _STEP is an Assembler subroutine that Steps the motor one step )
CODE _STEP ( -- ; uses X, D ;; Jumps to next MCB's action )
ASSEMBLER
    CURRENT_MCB ^ LDX

    ( Toggle Step Bit UP )
    STEP_BIT  ,X  A LDA
    A ASL
    >PORT ^  A EOR
    >PORT ^  A STA
    ( Toggle Step Bit Down )
    STEP_BIT  ,X  A LDA
    A ASL
    >PORT ^  A EOR
    >PORT ^  A STA

    >_WAIT ^ LDD
    ACTION ,X   STD

    CURRENT_MCB ^ LDX
    NEXT_MOTOR ,X LDX
    CURRENT_MCB ^ STX

    ACTION  ,X  LDX
```

tionary and dictionary header (and Forth assembler) see *Starting Forth* by Leo Brodie. Specifics of the Max-Forth dictionary can be found in the New Micros *Max-Forth Reference Manual Addendum*. A lot of what is written about Forth internals in this article is based on the "Max-Forth Reference Manual Addendum". CREATE <name of code word> or CODE <name of code word> simply put a header into the dictionary. When the <name of code word> name is interpreted, it puts the address of the next available byte in the Forth dictionary (when it was compiled) onto the data stack (This pointer is refered to the parameter field pointer or PFA in Forth lingo). The postfix Forth assembler starts laying down the opcodes at this address immediately after the creation of <name of code word>. These type of words end in either RTI for an interrupt return, or RTS if they have been called via JSR or BSR.

To Spin a Motor

The code in Listing 2. is our motor control code. The code is thoroughly documented. But at times documentation can only carry the cradle of understanding so far. Two concepts are fundamental in understanding this model of stepper motor control. Concept one is that a regular interrupt "heartbeat" propels this system. Concept two is that this system broken into states. Concept two leads to the definition of the Motor Control Block (MCB) and the factoring of states into little--quick subroutines; one of which is fired on each regular interrupt.

The Motor Control Block

The MCB is the fundamental data object in our system. It defines the next action to take on interrupt. This action is one of two events for our current application: stepping the motor or waiting. The sensor action points to a subroutine which looks at the sensors associated with this motor and performs a regular sensor update function. If an encoder was connected to this MCB's motor, the sensor action would update encoder ticks in a variable within the MCB. The MCB is where all of this motor's data is maintained. (Just because you don't see it doesn't mean it cannot be added). The address of the motor as connected to the NMIS 7040 module is maintained in the MCB, etc.

This MCB is in a linked list of

MCBs. With this model, each motor within the list maintains its own context or all of that data specific to this motor. Three motors can be run simotaneously, while processing other events--like a communications network with a master processor by this approach. The model found here is the basic model used for asynchronicity in advanced robotics applications and device drivers. In the context of this list of MCBs there exists an Anchor MCB. This is the first and last MCB. The list is accessed via the Anchor. The first Motor MCB is referenced from the Anchor MCB. The Anchor MCB's action is *not* fired first. When all of the MCBs fire for this interrupt, the anchor MCB's action is fired. The Anchor MCB's action is an RTI; the way out.

You may seem a little confused now, especially if you have looked at the assembler words _STEP and _WAIT. Remember concept 2 from above. The motor run instance needs to be broken into states. The goal of this article is to spin the motor.In order to spin a motor the motor needs to be stepped. Two states are required to step the motor. State one _STEPS the motor, state two _WAITs for the motor to settle. Since this thing is based on the heartbeat or timed interrupt, the settle time must be controlled. (In the next article we will control the settle time to accelerate and decelerate the motor). If the motor were to step on each interrupt (32 milliseconds apart), the motor would chatter. This articles goal is to spin the motor--chattering will not do. While in the _WAIT state, each interrupt decrements this motor's TIMER found in the MCB. When the settle time has elapsed as per TIMER, the state is changed to _STEP and the timer is reloaded for the next _WAIT state.The next time around, the motor will be stepped, and the state set back to the _WAIT state.

The Next time around?

I hope the confusion is becoming less confusing.. Lets look at the sequence of code which completes each state.

```
CURRENT_MCB ^ LDX ( get currently )
                  ( active MCB in X )
NEXT_MOTOR ,X LDX ( get the ''next )
                  ( MCB to fire'' in X)
                  ( thru this MCB )
CURRENT_MCB ^ STX ( make the ''next )
                  ( MCB to fire'' the )
                  ( current MCB )
```

continued page 38

```
    0 ,X JMP
END-CODE
' _STEP CFA @ >_STEP !

( Waits a specific time between steps )
CODE _WAIT ( -- ; uses X, D ;; Jumps to next MCB's action )
ASSEMBLER
    PSHY              ( Save Max-Forth's data stack pointer )

    CURRENT_MCB ^ LDX
    TIMER ,X  LDY     DEY
    EQ IF
       DELAY_TIME ^ LDD   TIMER ,X STD
       >_STEP ^ LDD  ACTION ,X STD
    ELSE
       TIMER ,X STY
    THEN

    PULY              ( Restore Max-Forth's data stack pointer )
    CURRENT_MCB ^ LDX
    NEXT_MOTOR ,X LDX
    CURRENT_MCB ^ STX

    ACTION ,X LDX
    0 ,X JMP
END-CODE

' _WAIT CFA @ >_WAIT !

( The last MCB's -- Anchor MCB's -- action )
CODE LAST_ACTION   ( -- ; Returns from processing interrupt )
ASSEMBLER
    RTI
END-CODE

' LAST_ACTION CFA @ >LAST_ACTION !

( A stub for future sensor update actions;; i.e. encoders, etc. )
CODE SENSOR_ACTION   ( -- ; Must be called via JSR )
ASSEMBLER
    RTS
END-CODE

' SENSOR_ACTION CFA @ >SENSOR_ACTION !

( Output Compare Timer Interrupt handler routine Specific to Output        )
( Compare Timer 1 )
( This Interrupt Routine will process the list of MCB's.                   )
( The list must be circular!!! ANCHOR -> FIRST -> ... -> LAST -> ANCHOR    )
( The Anchor MCB's action must end with an RTI.                            )
( All other MCBs must get the next MCB's action and execute it:            )
( Effectively..                                                            )
(  ...  NEXT_MOTOR ,X LDX ; Get next MCB in list into X                    )
(       CURRENT_MCB ^ STX ; Hold onto this in CURRENT_MCB for next routine )
(       ACTION ,X LDX     ; Load a pointer to the Next Mcb's Action into X )
(       0 ,X JMP          ; Execute Next MCB's Action                      )
( Sets up to the next MCB )
CREATE ~MOTOR_HANDLER ( -- ; uses X, D ;; Processes list of MCB's )
ASSEMBLER
   OC1F # A LDA          ( Acknowledge Interrupt. )
   TFLG1 ^ A STA

   TCNT ^ LDD            ( Set Up for next interrupt )
   TIMER_OFFSET ^ ADDD   ( based on Interval from TIMER_OFFSET. )
   TOC1 ^ STD

   ANCHOR_MCB # LDX      ( Get Anchor )
   NEXT_MOTOR ,X LDX     ( Get First Motor from Anchor )
   CURRENT_MCB ^ STX     ( Set as first Motor )

   ACTION ,X LDX         ( Do First Motor's Action )
   0 ,X JMP

( End of ~MOTOR_HANDLER )

: INITIALIZE_MOTOR ( -- )
    ( Connect last block to itself and connect last action )
    ANCHOR_MCB  ANCHOR_MCB NEXT_MOTOR + !
    >LAST_ACTION @ ANCHOR_MCB ACTION + !
    ( set the interrupt vector )
    ~MOTOR_HANDLER  >TOC1 VECTOR ;
```

Controlling Home Heating and Lighting

A Personal Embedded Controller System

By Jay Sage

Although most *TCJ* readers are familiar with my regular column on Z-System software, I actually got my start in microcomputing with controllers. Ever since editor/publisher emeritus Art Carlson began asking for articles on embedded controllers, I have thought about describing my very first computer project, an Intel 8085-based controller that has been operating the heating and lighting systems in my house for the past decade.

Background

In the late 1970s I was working in Raytheon Company's Research Division developing charge-coupled devices (CCDs) for analog signal processing. To facilitate the measurement of field-effect semiconductor devices, I designed a number of test boards that used analog integrated circuits (especially operational amplifiers) and small- and medium-scale digital logic circuits.

At the time, Raytheon sponsored some Summer short courses at Cornell in various electrical engineering subjects. One of them was an introduction to the then new microprocessor chips. In return for its support, Raytheon was allowed to send one employee to attend the course. I was offered the opportunity; it sounded like fun, so I accepted. I enjoyed that course so much that on my return I started immediately on the design of a microprocessor-based system that would operate the electrical circuits in our house.

Why did I choose that application? A couple of years earlier, our house had been broken into, and to discourage burglars from a repeat performance, we put clock timers on half a dozen of our lamps. It was quite a nuisance, however, when a timer would turn off the only light in the room and we would have to crawl around on the floor in the dark trying to find the timer. The computer controller would make life easier by working in parallel with the regular light switches.

In addition, we were still in the aftermath of the 1973 oil crisis, and I thought we could greatly reduce our oil consumption by having the computer controller operate the electrical circuits in the heating system. Ultimately, this is where the system has really been productive.

It is going to take a couple of columns (at least) to describe this whole project. This time I am just going to lay the groundwork; next time I will get into the details of the computer circuit. Although the circuit was designed and built more than ten years ago, and I would do a number of things differently today, the basic concepts are, I think, still sound and instructive.

The Basic Functions

The functionality of the controller that is visible to the user is basically a flexible scheduler. It stores the day and time when actions are to take place. It also knows about various conditions, such as whether we are at home or away and whether it is an ordinary day or a vacation period. Scheduled events can depend on combinations of those conditions.

Two kinds of actions are supported: (1) turning electrical circuits on or off and (2) setting the thermostat for the heating system. Electrical circuit switching is actually controlled in two ways. The obvious way involves turning circuits on and off at specific wall-clock times. A second stored schedule turns circuits on and off at times relative to the onset of darkness outside, as determined by monitoring a photocell. I generally turn lights on based on outside brightness and turn them off based on wall-clocktime. In this way, the schedule does not have to be revised with the change of seasons. Also, lights will turn on automatically during dark rainstorms (and off again when the storm clears).

We will share more specifics about the operation of the controller next time. For now we will continue with a discussion of the internal control strategy used to operate the heating plant.

Heating Control Concept

Our house, like many in New England, uses an oil-fired-boiler to heat water that is circulated by a pump through radiators (cast iron in our case). A thermostat senses the air temperature and turns on the circulator when the room air drops below the set temperature.

The temperature of the water in the boiler is similarly regulated by what is called an aquastat. While the thermostat could be set back at night, either manually or automatically, the aquastat always has a fixed setting. In order to ensure adequate heat on the coldest day possible, the water temperature is generally kept at a very high setting, perhaps 180 degrees Fahrenheit (80 Celsius). Day and night. Warm weather and cold.

And what does all the hot water in the boiler do while it is sitting around waiting to be circulated to the radiators? Well, for one thing, it leaks its heat into the basement. Our basement being unfinished, this is nearly a complete waste. Second, it sets up a nice convection up the chimney. This literally sucks heat up the chimney, cooling off the water in the boiler and drawing fresh cold air into the house. More waste!

Besides being inefficient, this control strategy also provides less comfort. After the air in the house cools, the circulator comes on, bringing very hot water into the radiators, which then transfer their heat into the air. Soon the room has

warmed up and the circulators shut off. But, the radiators are still nice and hot, and they keep pouring heat into the room. The room gets much hotter than the thermostat setting, and during the long cooling down time with no heat, the rooms tend to have cold drafts. Not very nice!

The best thing to do, I thought as a physicist, was to control the water temperature so that the radiators were at just the right temperature to transfer heat into the air at the exact rate heat was leaking out of the house. The circulators could run continuously. On warmer days and at night, when the thermostat setting was low, the boiler water would be quite cool, and the losses, both into the basement and up the chimney would be much less. Also, there would be almost no swings in air temperature, and the steady low-level heat from the radiators would inhibit drafts.

Simple, linear heat conduction theory implies that the boiler temperature would have to exceed the desired (and actual) air temperature by some factor times the amount that the air temperature inside is to exceed the air temperature outside, that is

```
( Tb - Ti ) = K * ( Ti - To )
```

or

```
Tb = ( K + 1 ) * Ti - ( K * To )
```

where Tb is the boiler temperature, Ti the inside air temperature, and To the outside air temperature. The factor K depends on the amount of insulation in the house and the effectiveness of the radiators (big cast iron radiators are great; baseboard units are pretty poor).

More Sophistication

Once one has a microprocessor running things, one might as well really put its processing power to work. I, therefore, applied some more sophisticated control strategies on top of the basic one described above.

If the aquastat is set exactly at the temperature required to maintain the steady-state air temperature desired, then there will be no excess capacity to allow the house to warm up when the thermostat setting is increased. Therefore, an additional correction is included in the aquastat setting in proportion to the difference between the desired air temperature and the actual current air temperature. The equation thus becomes

```
Tb = (K1 + 1)*Ti - (K1 * To) + K2*(Ts - Ti)
```

where Ts is the thermostat setting.

Adding this term has several benefits. It allows the house to warm up during programmed increases in temperature (Ts-Ti > 0), as we just said. It also allows the boiler to remain extra cool during programmed decreases in thermostat setting (Ts-Ti < 0). Finally, it enhances the stability of the system against errors in the control parameters.

Cycling an oil burner on and off is not good for efficiency. During the first several minutes of a burn, while the combustion chamber (fire box) is still cold, the oil does not vaporize well and, therefore, does not burn as well. There is less heat and more soot, which builds up on the heat exchanger and impedes heat transfer into the water.

The best thing would be to have a variable-firing-rate nozzle. Then one could keep the flame on continuously and regulate its heat production to match exactly the leakage rate from the house. Today there are gas-fired boilers that work somewhat this way. As far as I know, this has never been done with oil. I suspect that burning oil is a tricky process and that nozzles are carefully optimized for a particular flow rate. Building an efficient variable-flow-rate nozzle may not be feasible.

My compromise was to use a nozzle that provided just a little more heat than would be needed on the coldest possible day. Experiments showed that I could change to a nozzle with only half the flow rate of the one that had been in the boiler! With a small nozzle, firing times are longer, and the inefficient burn time is a smaller fraction of the total burn time.

Naturally, we want the house to be quite cool at night, and there is no sense keeping the boiler warm during that time. Therefore, the controller is programmed so that when it sees a decrease coming in the desired air temperature, it automatically decreases the aquastat setting and allows the radiators to extract as much heat as possible. Typically, by the time we go to bed at night, the boiler is down to only about 25 degrees C, barely above room temperature.

This in turn introduces a new problem. With the boiler so cold overnight and such a low firing rate, if the system waited until the thermostat setting came back up to turn on the boiler, it might be many hours before the house would catch up. The controller is, therefore, made to look ahead and to compute the time to begin firing so that the house temperature reaches a future programmed temperature at exactly the time it is programmed for. On a really cold day, the boiler begins firing three or four hours ahead!

Normally the boiler is cycled on and off in response to the computed aquastat setting. If one used a single temperature set point, of course, the oil burner would turn on and off frequently, so, as with standard mechanical thermostats and aquastats, some hysteresis is included to stabilize this. In other words, the turn-off setting is slightly higher than the turn-on setting.

In addition, since the early firing of the burner is less efficient, the controller is programmed to give a minimum burn time, once firing has started, independent of aquastat setting. I figured that it was better to have the water slightly hotter than necessary sometimes rather than have short burn times. Water condensation in the cool chimney and boiler parts could lead to corrosion. The minimum burn time was chosen to ensure that the boiler flue temperature would reach at least 100 C.

There is a worry one always has with a circulating water-heating system--freezing. If there is a pipe near an outside wall, and if the circulator is off for a long period of time, and if the weather is very cold, a section of the pipe may freeze and possibly burst. This makes for quite a mess (especially if you haven't disabled the automatic water filling system, as I have). To prevent freezing, the software turns on the circulator for a short period of time after a maximum off interval even if no heat is needed.

As I mentioned earlier, with the control strategy implemented here, the circulator can run nearly continuously. In fact, that is exactly what happened with an early version of the software, and we noticed a substantial jump in our electric bill because of the energy consumed by the motor.

I then reasoned that it should be adequate to turn on the circulator only long enough to exchange the slug of water in the radiator for a new, warm slug of water. So long as the

continued page 36

Getting Started in Assembly Language

Making the Jump from High Level Languages

by A. E. Hawley

Why would anyone who programs in a high level language (HLL) want to learn assembly language, especially considering its reputation as "difficult and too detailed?" One reason is precision, another is power. Given programmers with equal skills, a program written in assembly language will be both faster and smaller than one written in a high level language. Can you think of a way to write a program which will load and execute at the highest available location in memory without disturbing one in low memory? You can do it with a good assembler and linker. It is very difficult to do in most high level languages.

John Poplett, as sysop of the "Plaster of Paris BBS", catered mainly to writers and literature buffs. John himself was a technical writer who had learned BASIC, gotten acquainted with ZCPR3, and had set up his BBS with the venerable RBBS system. He was a frequent caller on my Z-Node, always eager for the latest utility updates. We chatted often about the workings of the CP/M operating system and ZCPR3. One day, he decided that he needed to be able to understand in much more detail how things worked. He wanted to learn AL (Assembly Language) and wanted some advice on how to go about it. He picked a goal, and six months later the first version of ZLUX was released.

This article includes the advice that John Poplett used in his climb from "rags to riches" (intellectually, at least). It is introductory, intended for the average user familiar with one or more high level languages. Perhaps you, too, want to understand AL programming.

What do you need to get started? You probably already have some of the software lurking in your archives. You will need an editor, an assembler, and a debugger. Of course, if you never make misteaks you have no need for a debugger, wright?

Choosing an Editor

If you have done HLL programming, then you have an editor and will prefer it. The main requirement is that the editor be used in a mode that produces straight ASCII files. Assemblers just can't cope with the meaning of formatting commands and embedded control characters. Many programmers use Wordstar in the non-document mode. An excellent public domain editor that uses the Wordstar command syntax is ZDE, by Carson Wilson. ZDE is much faster than Wordstar because it does not page its files to and from memory. Wordstar (or some other) must be used if your file becomes too large to fit into available memory. I started years ago with MATE, graduated to PMATE and then to ZMATE. If you need to buy an editor for programming use, ZMATE is the one I recommend.

Assemblers

The assembler you use should be a relocating macro assembler that accepts Z80 mnemonics and can produce HEX and Microsoft REL output files. Bruce Morgen (reference 3) compares assemblers and linkers from DRI, Microsoft, and SLR. The DRI RMAC assembler, as he points out, uses Intel 8080 mnemonics but is otherwise acceptable. Through use of the public domain Z80.LIB, RMAC can be made to produce Z80 code but at the expense of non-Zilog mnemonics. The differences make your source file incompatible with assemblers that use Zilog mnemonics. If you use RMAC, be aware that you will have some translating to do when you try to assemble files (from the public domain, for example) written in Zilog mnemonics.

Bruce did not include the ZMAC assembler and ZML linker in his review because they were released a year later. ZMAC uses Zilog mnemonics, is CP/M compatible, uses ZCPR3x if it is present, and has a superset of M80 capabilities. It is the only Z80 assembler that is currently actively supported. This assembler handles HD64180 as well as Z80 instructions. ZMAC is much faster than M80 and about the same speed as the SLR assemblers in their two-pass mode. The ZML linker makes the generation of PRL and SPR files as simple as a command line option and can easily produce RSX files. Similarly, making ZCPR34 Type 4 files is a function that you simply invoke. Type 4 files execute at the highest possible memory address, just below ZCPR34.

If you have ZAS/ZLINK, go ahead and use them until you can get something better. They have some "features" that will eventually make you wonder if you really understand assembly programming! (be assured, you do..)

I have all of the above. Naturally, I

A. E. (Al) Hawley started out as a Physical Chemist with a side line love of electronics when it was still analog. He helped develop printed circuit technology, and contributed to several early space and satellite projects. His computer experience started with a Dartmouth Time-Share system in BASIC, FORTRAN, and ALGOL. His first assembly language program was the REVAS disassembler, written for a home-brew clone of the Altair computer. As a member of the ZCPR3 team, he helped develop ZCPR33 and became sysop of Z-Node #2. He has contributed to many of the ZCPR utilities, and written several. He is author of the ZMAC assembler, ZML linker, and the popular ZCNFG utility.

use ZMAC most of the time. Z80 source files written for M80 can be assembled by ZMAC or the SLR assemblers. I reserve RMAC for those public domain programs written for it when I don't want to bother with translation to Z80 mnemonics.

Debuggers

Bruce Morgen also reviewed debuggers. He reported DSD as the premier Z80 debugger. I have it, use it regularly, and agree that there is nothing finer in its field except its big brother Remote DSD. DSD is indeed available. Sage Microsystems East can supply it, as well as ZMATE and ZMAC/ZML.

You will find a symbolic debugger program very useful. Symbolic debuggers use the symbol table produced by linkers to allow you to single-step through the executing code, displaying the state of all registers and memory contents in terms of the symbols you assigned in your source code. A good public domain debugger, available on many remote access systems, is Rick Surwilo's Z8E. This symbolic debugger, like DSD, is screen oriented, versatile, and of professional quality. Cam Cotrill, one of the authors of ZSDOS, used Z8E on many software development projects until he discovered DSD.

Things You Need to Know

Now you have the basic software tools, and you have read the instructions provided with them. (You *have* read them, haven't you?) What else is there? Assembler documentation usually assumes that you are familiar with the basic mathematical, logical, and physical principles on which the operation of your computer is based. This awareness is the guide that you use to select and organize the instructions that comprise an AL program. Here's a list of subjects that are important:

How a CPU works
- Architecture for Z80, Z180/HD64180
- Data, Address, Control lines
- Machine commands and Machine language programming

How information is organized in a computer
- Bits, Bytes, Words, Addresses, and Memory organization

How data is manipulated
- Binary arithmetic--Add, Subtract, Multiply, Divide, and SHIFT
- Boolean algebra--AND, OR, XOR, NOT and Truth tables

Assembly language programming
- Assembler Mnemonics
- Assembler Pseudo-ops. How they differ from Mnemonics

How a CPU Works

The final answer to questions about how a microprocessor works is contained in the manufacturer's data sheet. References 1 and 2 point you to the right phone numbers to call to get the data sheets for the Z80 and its newer brother the HD64180 (Z180, from Zilog). Some electronic supply houses can also supply these. In either case, be pleasantly surprised if they are free. You should have one of these in your library, depending on the CPU your computer uses. Books on programming contain the same information, paraphrased and edited.

Machine Language Programming

The first widely available microcomputer was the Altair. It had a front panel with rows of LED lights and switches. The LEDs monitored the CPU control bus, the data bus, and the address bus. Sixteen of the switches were used to specify the address for the next operation. Eight of these switches provided data for the data bus. Other switches controlled running, stopping, single stepping, and storing data bus contents at the current address. With this primitive setup, the user could store bytes contiguously in memory to form an executable program, then execute the program in single step or run mode. Typically, the very small program so tediously entered would simply read bytes from an I/O port and deposit them in memory. When done reading, a jump instruction caused the new block of code to be executed, and voila! BASIC, or a monitor, or simple editor/assembler came alive.

Your computer starts up the same way! The front panel has been replaced with two switches: the power switch, and the Reset button. The initial bootstrap code is in ROM, automatically invoked when power is turned on or the reset button pushed. With all debuggers, you can simulate the original method of programming without getting calluses on your fingers. You use the "S" command to enter bytes where you wish in memory. You specify the bytes in hexadecimal, but some debuggers allow you to specify them in binary or ASCII as well. Then you can use the "GO" command to execute the code, or a "T" to single step through the code. At any point, you can observe the contents of the CPU registers and memory locations. This is the first step toward a high level language. An assembler is the second step.

Bits, Bytes, Words, and Addresses

Your computers CPU processes data using byte-based binary boolean logic operations, shifting operations, and copy operations. Integer arithmetic operations are actually microprograms within the CPU, and are executed like the examples of binary arithmetic you can do by hand (the long hard way). Single-bit operations can be visualized as boolean byte operations in which bits other than the bit being addressed are either unaffected or are all set to either 0 or 1.

The CPU logic operations provided are AND, OR, XOR, and NOT. The NOT operation masquerades in the CPL instruction, which simply inverts all the bits in a byte. The arithmetic operations are ADD and SUBtract, with variations depending on the treatment of the carry bit and extension to operations on 16 bit words.

If you are at all hesitant about these subjects, by all means look to books like references 7,8, and 9. The Intel *8080 Assembly Language Programming Manual* (reference 10) discusses the arithmetic and logic instructions in great detail, illustrating them clearly in binary. The Zilog and Hitachi publications assume that you are already familiar with such basic theory. The owners manual for the Altair computer (if you can find one!) contains a clear discussion of these principals although, of course, it is limited to 8080 instructions. The Z80 had not yet been developed!

From Machine Language to Assembler

In the first step toward higher level programming manual operations (flipping switches) are replaced in the debugger by much less tedious typing of hex bytes. A second level of abstraction occurs when you use the simple in-line assembler that most debuggers contain to enter the mnemonics for machine instructions (like LD A,0). Note that numeric values

continued page 37

REAL COMPUTING

The PC-532, Minix 1.5, and the 32GX320

By Richard Rodman

My PC-532 is up!

The basement is small, but that's no obstacle to its looking like the lab from Bride of Frankenstein. A web of cables blanket the floor. The patient lays atop a makeshift table. Meters and probes are precariously balanced over the exposed circuitry, bathed in the cold phosphorescent glow of the monitors. He sweats as he tirelessly moves his probe from point to point-no signal, no signal, no, ... "More POWER! I need more POWER!"

A sudden flash of lighting silhouettes him against the casement windows. An LED flickers! The monitor stirs! He jumps up and throws open the window. "It's ALIVE!" he screams into the dark rain that pelts him in the face. "It's alive." He stands, laughing and sobbing, soaked by the rain, as the lighting flashes, the thunder booms.

What can match that supreme joy, that feeling of sublime affirmation, that incredible, indescribable pinnacle of achievement when the object of one's simultaneous boundless love and bottomless hatred, one's most enticing temptress and fiendish enemy, after hours, days and months of unflagging pursuit, untold perils and changes of strategy, finally gives in and works!

What I'm getting to is that my PC-532 is now up. It was actually a struggle more against ambiguous documentation than anything else. For one thing, there are 8 SIMM sockets. To install only 4 SIMMs, you use the even-numbered (every other) sockets. For another thing, the console connector is J3. The documenter must have assumed that these facts were too obvious to merit inclusion in documentation, but he forgot the first law of documentation: Nothing is obvious.

Anyhow, the PC-532 has the greatest ROM monitor I've ever seen. It was written by Bruce Culbertson and includes--get this--on-line help! In these days of cheap ROM, it's about time. The monitor supports breakpoints, single-step, instruction disassembly, and a lot of other nice features. It includes a simple binary download protocol. It was a simple matter to modify my HOST program to support this protocol too. Using this downloader, following Bruce's instructions, I was able to set up a RAM-disk and bring up Minix 1.3 on the PC-532. I don't have any disk on the system as yet, however.

By the way, a couple of issues back I mentioned that the OMTI 5200 board is available, but left a funny symbol as to who it was available from. Actually, this is an older controller. A newer model, the 7200, is available from Arrow Electronics. There are some variations in different models. The model 7200 can interface up to 4 floppies and 4 MFM hard disk drives to the SCSI bus.

Getting back to the monitor, it's made up of a combination of assembly and C. Bruce has been using GCC (the Gnu-C compiler) which he has running under Minix. I haven't been able to rebuild it as yet using the tools I have available, but would suggest that this monitor would be better than SRM or TDS as a standard for home-brew NS32 systems.

Minix 1.5

Minix 1.5 is no academic exercise. This is a full-featured operating system, with more system calls and utilities than version 7 Unix. It runs quickly and smoothly, supports all the various floppy formats of the PC, and is well documented. It's actually even better than I expected it would be. It came, in my case, on 12 720KB 3-1/2" floppies.

It's almost amazingly audacious to come out with a program available on the PC, the Atari ST, the Amiga and the Mac, all at the same time--but an operating system! It's a potential customer support nightmare. A mere word processor has a cushion of device-independent code under it; an operating system has to work right on the bare metal, and some pretty quirky bare metal it is, too. But they're doing it; call them at the numbers below if you have trouble. Obviously, OS hackers are made of better stuff than your average 1-2-3 user. It appears from Usenet that Mac users have had some difficulties running Minix under Multifinder. I had two bad floppies, which Prentice-Hall replaced cheerfully. But, at least on the PC, this is a very stable product.

To misquote Stan Kelly-Bootle, into each china shop some bull must fall. I had some problems with Minix 1.5. After going though 8 pages of installation procedure in which, all along, they assure you it can be installed with the hard disk as the root (not using a RAM-disk), the setup_usr shell script cannot dismount the hard disk, so it won't execute, and the software can't be installed. This means that you have to start completely over and re-partition the hard disk. Remember: you must use a RAM-disk.

On the bright side, Minix's fdisk program is tremendously better than DOS's. Having used it, I wonder how Microsoft is getting away with nationwide distribution of such a shlock partitioning program. DOS's FDISK is cumbersome, limited, confusing, error-prone, and clunky, and hasn't been improved at all since they came out with it 5 years ago. They ought to be ashamed.

Minix also includes a network driver for the Amoeba protocol. It includes drivers for a couple of common Ethernet boards. This means that networks of Minix systems are now possible. And, of course, source is included. About all you don't get is X window. The way things are going, someday maybe AT&T will brag about having a Minix-like operating system.

More information on the 32GX320

I've received some more information on the new processors in what is now called the Series 32000/EP. First I'd like to go into what I think is the more amazing of the new chips, the 32GX320.

As mentioned before, this chip is a superset of the 32GX32. It does not have, nor support, a memory management unit. It does have an on-chip interrupt controller, DMA controller, and some new instructions. It also has some hardware assistance for existing instructions.

The on-chip interrupt controller is not as fancy as the 32202, but it has what most people need. It includes 3 16-bit timers. The logic is mapped into memory at FFFFFE00 and at FFFFF800 (don't you love these 32-bit addresses?). The timers can count external signals and can also generate system timing outputs. The data book (it can hardly be called a "sheet") notes that "from the CPU standpoint, the on-chip ICU can be regarded as an independent module."

The DMA controller has two identical and independent channels. Each of these are capable of operating in several modes. The fastest mode is what is called "flyby" mode. In this mode, both an I/O device and memory are selected simultaneously, and the data is transferred in a single bus cycle. DMA controller channels may also be used for transferring memory blocks within the system. The DMA controller is mapped into memory at FFFFF010 and up.

How about the new instructions? The first one is MULWD, a 16x16 multiply which is functionally the same as MULW, but around three times as fast (200ns). MULW is still available if you want to use it. The next is CMULD, complex multiply double. The two operands of this instruction are 16 bit real part in the low-order word, and 16 bit imaginary part in the high order part. The result (32 bits each real and imaginary) does not replace either operand but goes in registers R0 and R1. Another complex math instruction is CMACD, complex multiply and accumulate double, which is like CMULD except the results are added to the previous R0 and R1 contents. Finally, there's MACTD, multiply and accumulate twice double, which is a kind of vector dot product.

Shift instructions on the 32GX320 are sped up by a barrel shifter. There's no new instructions, though; the normal instructions just run faster.

The chip is available either in a 208-pin flat pack or in a 175-pin PGA. No, it's not pin compatible with the 32532 or the 32GX32. The data book number is item 114303. This is a hot chip. Let's hope National can get a bunch of these into the hands of experimenters out there.

Next time

Next time, we'll have the details on the 32FX16 and a couple of other fax-related products. Plus, the old dog's next new trick: the 486 has a crude cache. In the meantime, when you hear someone describe their product as "open", keep your eyes open and your wallet shut.●

Where to write or call

Minix: Prentice-Hall	Voice:	1-800-624-0023
Microservice Customer Service		1-201-767-5969
Simon & Schuster	BBS:	1-703-330-9049
200 Old Tappan Road		
Old Tappan, NJ 07675		

William P Woodall · Software Specialist

Custom Software Solutions for Industry:
- Industrial Controls
- Operating Systems
- Image Processing
- Hardware Interfacing
- Proprietary Languages
- Component Lists

Custom Software Solutions for Business:
- Order Entry
- Warehouse Automation
- Inventory Control
- Wide Area Networks
- Point-of-Sale
- Accounting Systems
- Local Area Networks
- Telecommunications

Publishing Services:
- Desktop Systems
- Books
- CBT
- Format Conversions
- Directories
- Interactive Video

33 North Doughty Ave, Somerville, NJ 08876 · (908) 526-5980

Z-SUS...

Z-System

Software Update Service

Bringing the Z-System World Closer Together.

Once upon a time, before the coming of the Corporate Market, microcomputers were not just owned by individuals. They were built and programmed by individuals. We were an ignorant lot back then. We never thought to ask if a machine could do something--we just plugged away at the task until it did! Those innocent days are gone forever, replaced with $500 text editors, exploding windows in gaudy colors and simple utilities expanded past 100k in object form. There is no room for the individual anymore.

Or is there?

Z-SUS...

Bringing the Z-System World Closer Together.

Reenter the exciting world of truly personal computing. The cost is small: a Z80 computer, a new operating system and time. Learn what others have never forgotten, that the spirit of the individual made this industry.

Z-SUS Catalog Disk:

An in-depth catalog of *Z-SUS* offerings, available only on disk. Besides providing you with a detailed explanation of every service and package we offer, the *Z-SUS* catalog disk also includes the most current copy of ZFILEV.LST by Bill Tishey. To this, we've added Gene Pizzetta's XFOR.COM, adapted and renamed to DF.COM (Describe Files). Use it to look up the name of any Z-System tool by any criteria you wish. For example, to find files relating to shells, you would enter:

```
A0>DF SHELL
```

Now you can quickly search for just the tool you need to solve any problem at hand.

Price:	$ 2.00
Foreign Shipping Surcharge[1]:	2.00

Z-SUS Subscription:

The guaranteed way to stay up-to-date with the latest public domain Z-System software. Most include source code. The subscription disks are issued approximately once per month. Start your subscription at any point you like. Just tell us on your order form where to begin.

We are always looking for good software. If you know of a release deserving worldwide distribution, send it in!

Price, 12 issue subscription:	$72.00
Price, 6 issue subscription:	48.00
Foreign Shipping Surcharge[1]:	24.00 (12 issues)
Foreign Shipping Surcharge[1]:	12.00 (6 issues)

Z-SUS Disks, ordered individually:

These are the disks that subscribers receive approximately once a month. Order just the disk you want! Or put in a subscription to begin on any issue to receive that issue and the following 5 or 11 (depending on length of subscription you select).

Price per issue (1 to 3 issues):	$10.00
Price per issue (4 to 9 issues)[2]:	9.00
Price per issue (10 or more)[2]:	7.50
Foreign Shipping Surcharge[1]:	2.00 per disk

Z3COM Package:

A full set of executable (.COM) files supporting ZCPR3 and the Z-System. The package consists of 456 files totaling 1,564+k, as of January 1, 1991. It comes on six DSDD disks in a collection of libraries. The *Z3COM Package* contains earlier programs designed to run under ZCPR 3.0 and 3.3 as well as the newest utilities supporting NZCOM and ZCPR 3.4. It includes some essential general CP/M programs such as ARK, UNARC, CRUNCH, UNCR and NULU as well. You will be amazed at the treasures to be found in this collection!

Price:	$36.00
Foreign Shipping Surcharge[1]:	4.00

Z3HELP System:

The *Z3HELP System* consists of a set of libraries containing on-line help for nearly all available Z-System programs and utilities. It is designed to work with LBRHELP. Examples are given to assist you in installing this comprehensive system. Updates are published in each issue of the *Z-SUS* subscription disk.

At the present time, the *Z3HELP System* contains well over 1 Megabyte of compressed files and is distributed on 4 DSDD disks. This is one of our most popular packages.

Price:	$20.00
Foreign Shipping Surcharge[1]:	3.00

TCJ Articles:

A collection of articles written by Jay Sage, Bridger Mitchell and others relating to Z-System as printed in *The Computer Journal.* We encourage you to subscribe to *TCJ*, the only general circulation magazine offering strong support to CP/M and Z-System, and offer this collection to show the quality of articles. Distributed with permission on two DSDD disks.

Price:	$10.00
Foreign Shipping Surcharge[1]:	2.50

And the Best New Software Closer to You!

And let us help guide the way. *Z-SUS* keeps Z-System users around the world updated on the newest public domain software--the heart, and soul of personal computing.

Z-SUS Programmer's Pack:

A comprehensive collection of tools for the Z-System programmer. This set of 8 DSDD disks includes all the official Z-System Libraries, as well as essential utilities, modules and routines which today's Z-System programmers and developers find most useful. Both Microsoft and SLR formats of the official REL files are included. (Note: source code for the Libraries is proprietary and could not be included). You receive complete documentation, on-line help and the full release LBR files for all included tools: everything needed to join the ranks of Z Programmers!

Price:	$48.00
Foreign Shipping Surcharge[1]	5.00

Custom Order Disks:

Need something from the past? If it is listed in ZFILEV.LST, then the *Z-SUS Custom Order Disk* service can get it for you! Best yet, the price is ***per disk***, not per file! Just be sure not to exceed your disk capacity. Order full release LBR files only. Sorry, no discounts on this labor intensive service.

Price:	$10.00 per disk
Foreign Shipping Surcharge[1]:	2.00 per disk

Z-SUS Word Processing Toolkit:

This package contains Z-System programs and utilities useful in text and word processing. Classics such as Carson Wilson's ZDE are included as are tools and tips for handling and printing text in its many forms. You even get a full-blown spelling checker! Over a megabyte of software distributed on four disks. Special thanks to Carson Wilson, Gene Pizzetta and Lee Bradley for input to the selection of files for this package.

Price:	$20.00
Foreign Shipping Surcharge[1]	3.50

A Note of Recognition:

The many files that *Z-SUS* offers you are the result of thousands upon thousands of hours of effort on the part of hundreds of individuals. Each has donated his or her work to the public domain. It would be inappropriate to enjoy the fruits of their labor without expressing our gratitude. To all the folks who write for the Z-World, *thank you!* You are the very heart of Z-System.

Ordering Information

[1]***Foreign Shipping Surcharge:*** Prices shown are for orders to the US, Canada and Mexico. All others must add shipping surcharge. Sorry, surcharge is not subject to discounts as it reflects actual costs we incur in processing your order.

[2]***Z-Node Sysop Discounts:*** Z-Node sysops receive a 50% discount from standard rates on all *Z-SUS* products except *Custom Order Disks.* Multiple-disk rates on individually ordered *Z-SUS* subscription disks and foreign shipping surcharge not subject to discount.

Note to Amstrad Owners: Due to high cost of disks, a surcharge of $5 per disk must be imposed. This is waived if you supply your own formatted disks in either 48-tpi or 96-tpi formats.

Formats Available: We support nearly all 5.25 inch CP/M formats, both 48-tpi and 96-tpi, as well as 8 inch. Please inquire regarding hard sectored formats.

Method of Payment: Payment must be in US funds drawn on a US bank.

Z-SUS

What's It About and Who's Behind All This, Anyway?

The *Z-System Software Update Service (Z-SUS)* is a reincarnation of an old idea. Years ago, when the Z-World looked to Echelon for products and support, it was recognized that the fastest way to keep users up to date was through use of modems and remote access computer systems dedicated to the task. Thus was born the network of Z-Nodes.

But it was also recognized that not everyone had modems. And some who did lived in remote areas, or outside the United States. For these people, a simple call in to a Z-Node was not simple at all! At the least, it was prohibitively expensive.

Echelon established a mail order service to keep such users updated. But that was then. In the interim, Echelon went the way of so many other companies: into the drink. Left behind was a world of Z-Users who still needed support.

The Z-Nodes remain. In fact, should you have occassion to call the typical Z-Node, you will find the activity is greater than ever. However, the important task of supporting the remote and non-US user remained unresolved.

In the summer of 1989, Chris McEwen, sysop of Socrates Z-Node 32, approached Jay Sage with the idea of reviving the update service. As the author of the last two versions of ZCPR, and a Z-Node sysop himself, Jay was very interested in the proposal. Both recognized the need but neither felt capable of handling the task of compiling the necessary software or editing the issues. For months it seemed the new *Z-SUS* would be still born for lack of an editor.

Meanwhile, Bill Tishey was busy on his own compiling lists of Z-System software releases and meticulously preparing libraries of help (on-line documentation) files. It was decided that Bill's skills were needed for *Z-SUS*. The question was, would he be willing to take on the tremendous task?

You have before you Bill's answer. *Z-SUS* today has a broader product list than ever before! Users from around the world rely on regular update disks and special packages provided through the system. You can, too.

Who, again, makes up *Z-SUS*?

Jay Sage is the publisher and business agent. He handles your account, accepts orders and keeps things on track. Jay operates Newton Centre Z-Node 3.

Bill Tishey is the editor. He searches the world for the latest and greatest public domain software releases for Z-System. From these, Bill compiles the regular *Z-SUS* subscription disks, and the many special packages. Bill welcomes your ideas for new *Z-SUS* products.

Chris McEwen handles production and distribution. Write to him regarding problems with disks. He also handles *Z-SUS* promotional efforts. If you know someone who needs our service, let Chris know. Chris operates Socrates Z-Node 32.

Z-SUS exists to serve you! Let us know of any ways we can serve you better.

Z-SUS Order Form

Name _____ Computer Type _____

Address _____ Drive Type / Capacity _____

_____ Format _____ Alternate _____

City _____ Telephone _____

State/Province _____ Zip _____

Item Description	Qty	Total
_____	_____	_____
_____	_____	_____
_____	_____	_____
Z-Node Sysop Discount[1]		_____
Foreign Shipping Surcharge[2]		_____
Order Total		_____

[1]Not Applicable to Subscription Disks ordered individually, or Custom Order Disks

[2]Applies to orders outside US, Canada and Mexico only

For subscription orders: Which issue should we start your subscription on?
Most Current Issue:_____ Volume: _____/Issue: _____

How are you paying for your order: Check:___ Money Order:___ Visa:___ MasterCard:___

Visa / MasterCard Account Number: __ __ __ __ / __ __ __ __ / __ __ __ __ / __ __ __ __
Issuing Bank:_____ Expires:__ __/__ __ Signature:_____

Payment must be in US Funds drawn on a US bank.

Mail your order to:
Sage Microsystems East
1435 Centre Street
Newton Centre, MA 02159-2469 USA
Voice: (617) 965-3552 (Sunday - Thursday, 9:00 am - 11:30 pm)
Data: (617) 965-7259 (24-hr, 3/12/2400 bps, password=DDT)

LAN Basics

By Wayne Sung

For several issues we will be talking about local area networks (LANs) in general and Ethernet specifically. Before diving in, I thought I would try to define some items commonly used in LANs. Sometimes I forget that not everyone is in the networking business.

The biggest difference between a LAN and a number of point-to-point connections is the sharing of the LAN by all devices connected to it. In contrast, each point-to-point connection is totally controlled by the owner of the port involved.

This sharing means that only one station attached to the network may transmit a message at any given time. The rules that govern this process constitute an access method. The access method for Ethernet is called CSMA/CD (carrier sense multiple access with collision detect).

The cabling that runs from one station to another is called the medium. This can be coax, twisted pair, or fiber. There are even some wireless LANs, using radio waves or infrared light. Ethernet started life as a coax medium but today runs on many different media. Twisted pair particularly is coming on strong.

Local area networks tend to run at fairly high speeds compared to point-to-point transmission. This is not universally true, though. Apple's Localtalk runs at 230.4 Kb/s, compared to 10 Mb/s for Ethernet. However, this network signaling rate is not the most important factor in choosing a LAN. More on this in the next issue.

Stations (typically computers) attach to the medium through transceivers. These are small boxes with connectors for the medium side and the computer side. In high-signaling-rate systems such as Ethernet these boxes are metal for shielding. Lower frequency systems, such as Localtalk, can use plastic cases.

In Ethernet, transceivers attach to the coax in a number of ways. In thick Ethernet systems, it is possible to drill a hole in the coax (thick coax is an inch in diameter) and mount a tap. The transceiver mounts to the tap.

In thin coax systems, the cable is usually cut and two BNC connectors are mounted on the cable ends. Then a T connector allows the transceiver to be connected.

The transceiver converts the separate input and output signals the computer uses to the single coax that all stations use. In Ethernet there is also a hardware collision detect function in the transceiver. The transceiver cable which connects to the computer has pairs for transmit, receive, collision detect, and power.

Often the transceiver is built directly into the network controller that fits in the computer. This eliminates the transceiver cable as well as the transceiver casing and allows significant price reduction.

Local area networks are often characterized by topology, that is, the type of picture that results when the network is drawn schematically. Thus Ethernet is often called a bus topology LAN, because it tends to look like a straight line with stations coming off the line. Another common topology is the ring, used in IBM Token Ring and others.

Note that this is a logical topology, and does not in any way dictate the physical topology. In running wire to connect the stations, a star shape (a central point with spokes radiating from it) is the most common. This is because most buildings already have ways to get wire from any room to a wiring closet.

When using fiber it is not possible to tap the line the way coax is tapped. This is because light does not split in a fiber the way current splits in a wire. When using twisted pair the rules call for each transceiver to be connected to a repeater individually. Thus in these two cases a star wiring scheme naturally results.

A LAN is best characterized as a switch which allows any attached station to send messages to any other. The format of the messages is very similar to a a piece of mail. There must be something to identify who sent the message and whom it is intended for. This is why these messages are often called packets.

The contents of each packet are independent of the delivery mechanism. Usually there is a hardware-generated checksum (CRC-32 in Ethernet) available for error-checking.

In actual use the exact type of LAN is not all that important. Correctly built, any LAN works well. However, bad practices abound and the tone of the my articles in subsequent issues of *TCJ* is a little cynical, because I have had to locate and fix many of these bad practices.

Much of what I will have to say is based on experiences gathered while engineering the coNCert network in North Carolina. (coNCert stands for Communications for North Carolina Education, Research and Technology. This is a data and video network).

I am primarily a hardware person, and as such am always on the lookout for measurement methods that can help me do my job. It is this search for test methods concerning LANs that led to the design of the first Gag-a-matic. But first some history.

In 1985, the job of building a statewide high speed data network began in earnest. The goal was to go to T1 data rates (1.5 Mb/s) across cities as soon as possible and go as high as possible after that. This was based on the design

Wayne Sung has been working with microprocessor hardware and software for over ten years. His job involves pushing the limits of networking hardware in attempting to gain as much performance as possible. In the last three years he has developed the Gag-a-matic series of testers, which are meant to see if manufacturers meet their specs.

of the microwave system used for inter-city communications. These were built for multiple T1s with a total capacity as high as 45 Mb/s.

At first Ethernet bridges were used to join the various campuses that were part of the network. These first ones couldn't actually run at a full T1. Over a year later, more bridges were acquired which could use a whole T1.

When the campuses were first linked, there were perhaps 250 computers involved. At first the growth was not staggering. Even two years after the initial links were established there were maybe 500 computers connected. Then things began to take off.

As the price of workstations began to decline, connections happened at a much more rapid pace. Today, just over four years later, there are more than 3,000 computers connected. The drastic increase in traffic volume uncovered some weaknesses in the network.

One of the things that showed up was that there was so much broadcasting and multicasting going on that machines were literally spending all their time processing these and not doing any useful work. This will be discussed in more detail in the next issue.

Even though we were trying to control this problem by deliberately cutting off some types of broadcasts and multicasts, it turns out they tend to increase geometrically with the number of machines connected, and we knew we had to go to a different way of connecting the campuses, namely routers. More on these in two issues.

Back to the subject of testing. At first there was a commercial LAN tester available to me, but I never liked it because it was not at all easy to set up, and often by the time I could get it running the problem had gone away. Thus I decided to see if I could do better developing my own test devices.

In most commercial testers, you get a packets-per-second reading. However, they almost always average this over a one second period. In an Ethernet, packets with minimum spacing between them and packets evenly spaced are very different loads, even if the total packets are the same. Closely spaced packets are much more difficult to receive.

In an interrupt-driven receiving device, there will be a minimum response time. If the peak packet rate exceeds what this response time allows, then packets will be lost. Even if the receiving device is not interrupt-driven, you might still lose packets because there aren't enough cycles to process packets at such a high rate.

You also cannot arbitrarily decrease the averaging interval in the measurement device, because the same number of instructions will need to run more often. This will cause the device to run out of cycles. As it turns out, this commercial device I had, even though represented as a full speed device, ran out of cycles anyway.

Rather than trying to build a faster software device, I looked at what I was trying to measure. It turns out to be the energy in the line. By integrating the voltage pulses on the line, I could get a much better picture of bursts. While trying different ways to make this measurement I tried putting an audio VU meter across the line.

This type of meter normally did the type of integrating I was looking for. In audio work you have the same problem to solve. A short loud sound is perceived differently by the human ear than a continuous sound. The VU meter compensates for these differences. I still use this meter for quick checks.

As the speed of bridges and routers improved rather quickly, we found that normal traffic could no longer outrun these units. At the same time, the advertisements (and sometimes even the specifications!) showed such high processing rates that it seems you could never overload them. Thus the Gag-a-matic 1 was born.

A friend of mine in the music business likes to turn amplifiers on full and hear what they sound like when driven to distortion. He would always say, "Let's gag it." Since I was doing the same thing to networking devices, I named my testers in honor of him.

In essence a Gag-a-matic tries to deliver Ethernet packets at as high a rate as possible. The first such was a Multibus Ethernet controller.

It wasn't long before we were testing Ethernet devices for throughput. Gag-a-matic 1 puts out over 14,700 packets per second (at a 64 byte packet size the Ethernet limit is 14,880). The packets themselves are limited to one or two different types, but when this unit was built we were using only bridges, and we didn't need too many packet types.

Few devices have succeeded in passing this stream. We quickly found out that our commercial tester simply could not keep up. By slowing the stream down we found that the tester started losing detail at as little as 3% load and beyond 10% was hopeless.

It turned out that the Ethernet board in the tester has about 700k of memory, and as long as that wasn't exhausted the packets coming in could be captured. Once this memory was used up, packets had to be moved to host memory. It took several packet times to move one packet, thus much data would be lost.

Since I only wanted packets-per-second, it occurred to me to put a frequency counter on the carrier detect pin of an Ethernet device. This is how Gag-a-matics are "calibrated", since if only one device is talking there are no collisions and the frequency counter produces accurate readings.

I now have Gag-a-matic IV (subtitled "Just when you thought it was safe to go onto the network again"). This is not the fourth unit but rather is a four-headed device, which can generate four different packet streams at the same time with enough CPU capability to calculate protocol-related data in the packets. Salesmen are beginning to hate us.

Interestingly enough, while G1 is a 10 MHz 68010, G4 uses four 33 MHz 68020 processors but actually is slightly slower -- it only does about 14,600 packets per second per stream. This is due mostly to the different Ethernet devices, but G4 is so much easier to load that I could actually bear to write a "user interface" to it.

On the other hand, G1 has no more than a debug monitor, and I have to load S-records into it, so I tended to keep the test programs as small as possible. G4 has ways of loading from Ethernet.

Nonetheless, I had to kick out the resident operating system when I started the Ethernet device. Without directly controlling the chip, the speed was maybe 1,500 packets per second. I was shown how to bypass most of the operating system, but even calling the Ethernet driver directly only got me up to 4,000 packets per second.

I wanted G4 because we finally convinced the manufacturer of the microwave system to give us a way to use the whole 45 Mb/s as one stream rather than 28 T1's. Thus I wanted to generate the equivalent of four full Ethernets of traffic. The devices that can switch that much traffic are a little hard to get as yet.

These articles are not meant to show how to build a LAN. Construction practices are usually supplied by component vendors, and should be followed closely. In the next issue we will examine some not-so-obvious things that might prevent you from getting maximum performance from your LAN.●

The Z-System Corner

Putting the NZCOM Virtual BIOS to Work

By Jay Sage

For this issue we are going to look once more at ways to use the NZCOM virtual BIOS. Ten issues back, in *TCJ* #39, I talked about how one can take advantage of NZCOM's incredible ability to change the BIOS, statically and dynamically, without requiring any source code for the computer's real BIOS. In that column I described how Cam Cotrill, one of the authors of the Z-System DOS (ZDOS), used the NZCOM virtual BIOS to overcome problems experienced with ZDOS on computers whose real BIOS failed to preserve the Z80 index and/or alternate registers across BIOS function calls, as is required for ZDOS. To Cam's code, I added my own enhancement to implement the Z-System environment's drive vector.

I had hoped that those two examples would inspire others to apply this technique to a wide range of problems, but so far nothing has appeared. It's not for lack of a need, however, for I have seen on Z-Nodes quite a few discussions of issues that could be handled quite nicely in this way. Ever the optimist, I will try a few more examples. Two of the examples will essentially be the solutions to problems asked about recently in messages posted on Z-Nodes.

The starting point will be the modified NZCOM BIOS that I discussed in the earlier *TCJ* column. The main parts of the code are shown in Listing 1. Several interesting features are worth pointing out before we proceed to our new modifications.

The beginning of the code has the material required for the generalized loading concept developed by Bridger Mitchell and described in detail in *TCJ* issue #33 (if you don't have all these back issues, you would do well to pick them up). The NAME statement embeds an ID into the REL object code. The NZCOM.COM and JETLDR.COM loaders use this ID to recognize the function of a module they have been asked to load so they can figure out its proper load address. The COMMON statements allow any addresses in the Z-System to be installed into the code by the loader at load time. This is the beauty of Bridger's concept: a single binary file can be used in many different systems.

The next section of the code must adhere to a rigidly defined format. First comes the table of jump vectors as required in the CP/M-2.2 specifications. With the exception of the cold and warm boot routines, these jumps would normally go directly to the real BIOS. Since we want to intercept this code and enhance its functions, we vector to other locations in the virtual BIOS code. Note that the jump vector table has room for 13 extra custom BIOS functions that might be implemented in the user's system.

The block of jump vectors is followed by some special data for NZCOM. First there is an ID string that can be used by programs to determine if an NZCOM version of Z-System is currently running. Then there is a data byte with the number of the user area where the command processor file is stored. In an NZCOM system, the command processor is loaded from a file and not from the system tracks of the disk. Finally, there is the space for the file control block for the CCP file. Then comes the replacement warm boot code that loads this file.

This is an area where changes would be made in a multi-user system (a number of such systems are around, including some manufactured by Televideo). Imagine that there are two terminals running on the system and each user wants to run a different configuration of NZCOM. Clearly, they can't both use the same NZCOM.CCP file. Therefore, each user has to be set up with a different name or user area for the CCP file. One place this change would have to be made is in the virtual BIOS code. I am not going to say any more on this subject, but I hope that some day we will get a *TCJ* submission dealing with the issues of implementing Z-System on a multi-user system.

Now we finally come to the internal BIOS function routines, such as ICONST and ICONIN. All of these entry points have code that loads the A register with the address offset in the jump table of the real BIOS and then calls the routine DOBIOS, whose function was described in detail in my col-

Jay Sage has been an avid ZCPR proponent since the very first version appeared. He is best known as the author of the latest versions 3.3 and 3.4 of the ZCPR command processor, his ARUNZ alias processor and ZFILER, a "point-and-shoot" shell.

When Echelon announced its plan to set up a network of remote access computer systems to support ZCPR3, Jay volunteered immediately. He has been running Z-Node #3 for more than five years and can be reached there electronically at 617-965-7259 (MABOS on PC Pursuit, 8796 on Starlink, pw=DDT). He can also be reached by voice at 617-965-3552 (between 11 p.m. and midnight is a good time to find him at home) or by mail at 1435 Centre Street, Newton Centre, MA 02159. Jay is now the Z-System sysop for the GEnie CP/M Roundtable and can be contacted as JAY.SAGE via GEnie mail, or chatted with live at the Wednesday real-time conferences (10 p.m. Eastern time).

In real life, Jay is a physicist at MIT, where is tries to invent devices and circuits that use analog computation to solve problems in signal, image and information processing. His recent interests include artificial neural networks and superconducting electronics. He can be reached at work via Internet as SAGE@LL.MIT.EDU.

umn in issue #39. Briefly, DOBIOS saves all the alternate and index registers, calls the real BIOS function, restores the registers, and then returns to the calling program.

If we want to incorporate additional functionality, this is the place for it. The code in Listing 1 includes the enhancement to the select-disk code to make it check the drive vector and return an error code if the drive is not enabled. This code, too, was discussed in detail in issue #39.

Changing the Logical Drive Assignments

There are times when you may not like the way the manufacturer of your computer has assigned the logical drives (those things you know as A:, B:, and so on). On my Televideo 803H, the hard drive has the two partitions A: and B:, while the floppy is C:. As an example, I have implemented a special NZCOM BIOS that swaps drives B: and C:, so that the floppy can be referenced as B: and the second hard disk partition as C:. I'm not sure why one would want to do this, but there might be reasons.

The ISELDK routine with the additional code is shown in Listing 2. The code is pretty simple. When the SELDSK BIOS function is invoked, the requested drive is passed in the C

Listing 1. The modified NZCOM BIOS source code that protects the Z80 alternate and index registers and that checks the ENV drive vector when selecting a disk.

```
; Program: ZSNZBIO
; Author:  Joe Wright / Cameron W. Cotrill
; Version: 1.2
; Date:    26 January 1989
; Derivation: NZBIO.Z80 version 1.5

; This version has been modified to check the drive vector
; in the disk select routine.

; ... copyright notice and comments

        NAME    ('BIOZ12')      ; NZCOM needs 'BIO' as first
                                ; ..three characters
        COMMON  /_ENV_/
Z3ENV:                          ; Address of ENV module
CCP     EQU     Z3ENV+3FH       ; Where CCP address stored
DOS     EQU     Z3ENV+42H       ; Where DOS address stored
DRVEC   EQU     Z3ENV+34H       ; Drive vector address

        COMMON  /_CBIO_/
CBIOS:                          ; Address of real BIOS
        CSEG

; ... section with equates omitted

; Beginning of NZBIO.  The header structure is absolutely
; crucial to the correct operation of NZ-COM.
;        --> DO NOT CHANGE IT <--

START:  JP      BOOT            ; Cold boot
WBOOTE: JP      WBOOT           ; Warm boot
        JP      CONST           ; Console status
        JP      CONIN           ; Console input
        ;... rest of jump table
        JP      IWRITE          ; Write
        JP      LISTST          ; List status
        JP      ISECTR          ; Sector translation
        DS      (30-17)*3       ; Room for 13 extra jumps

SIGN:   DB      'NZ-COM'        ; ID for NZCOM BIOS
USER:   DB      0               ; User area for CCP file
ZCFCB:  DB      1,'NZCOM   CCP',0,0,0,0
        DS      17

; ...auxillary jumps for IOP (omitted)

; END OF HEADER

; The following code is free-form and may be moved around

; ... coldboot code (omitted)

; Warm Boot Entry

WBOOT:
        ;... some of code omitted
        LD      DE,(USER)       ; Log into user area where
        LD      C,GSUSR         ; NZCOM.CCP is kept
        CALL    NZDOS
        XOR     A
        LD      (ZCFCB+32),A    ; Clear current record
        LD      C,OPENF
        CALL    NZFIL           ; Open NZCOM.CCP
        LD      HL,(CCP)        ; Load it at CCP

; Read NZCOM.CCP to (CCP) until end of file (code omitted)

; ... BDOS service routines (omitted)

; The following calls build a shell around BIOS calls and
; preserve the IX, IY, and alternate registers as required
; by ZSDOS and ZDDOS (and common sense).

ICONST: LD      A,6
        JR      DOBIOS

ICONIN: LD      A,9
        JR      DOBIOS

; ... similar code for other functions omitted

ISECTR: LD      A,48

DOBIOS: LD      HL,CBIOS
        ADD     A,L
        LD      L,A             ; Never a carry from this
        EXX                     ; Swap to alternate reg's
        LD      (HLP),HL        ; Save alternate registers
        LD      (DEP),DE
        LD      (BCP),BC
        LD      (IXREG),IX      ; Save index registers
        LD      (IYREG),IY
        EXX                     ; Swap back
        EX      AF,AF'          ; Save alternate PSW also
        PUSH    AF
        EX      AF,AF'
        CALL    JPHL            ; Do BIOS call
        EXX                     ; Swap to alternates
        LD      HL,(HLP)        ; Restore them
        LD      DE,(DEP)
        LD      BC,(BCP)
        LD      IX,(IXREG)      ; Restore index registers
        LD      IY,(IYREG)
        EXX
        EX      AF,AF'
        POP     AF              ; Restore alternate PSW
        EX      AF,AF'
        RET                     ; Return to caller

; Special routines are coded here.

ISELDK: LD      HL,(DRVEC)      ; Get drive vector
        LD      A,16            ; Get 16-drive into B
        SUB     C
        LD      B,A
ISELDSK1:
        ADD     HL,HL
        DJNZ    ISELDSK1
        LD      HL,0            ; Value for invalid drive
        RET     NC              ; Return if invalid drive
        LD      A,27            ; Otherwise, use CBIOS
        JR      DOBIOS

; ... area for saving register contents (not shown)

; End of NZBIO
```

register. The code checks sequentially for values of 2 (C:) and 1 (B:), and it changes the value in C to 1 and 2 respectively. If the value is neither 1 nor 2, then the value is left unchanged. Finally, the BIOS routine is called. Thus, virtual BIOS tricks the real BIOS, which still knows the drives by their original names.

Once this code has been entered, say under the name SWAPBC.Z80, then it is assembled to a REL file (Z80ASM SWAPBC/R in the case of the SLR assembler). If you wish, the file can be renamed from SWAPBC.REL to SWAPBC.ZRL just to make it clear that it adheres to the ZRL standard. The loaders don't care which name is used. Now you just enter the command JETLDR SWAPBC.ZRL and, presto, you have a new BIOS with the drive designations reversed.

Some cautions are in order. Drive references that occur within the real BIOS will still use the same physical units and will not know about the swap. External routines that call the BIOS or DOS will see the drives as swapped. Thus swapping the A: drive (assuming that is where NZCOM.CCP is loaded from) will cause the CCP file not to be found. I just tried modifying the code in the listing so that it swapped A: and B:. I also changed the number 1 in the first byte of the NZCOM.CCP file control block to a 2. Now the CCP file will be loaded from the B: drive, which used to be the A: drive. It worked just fine!

In this example, the drives are relogged automatically by the loader. If you try writing a more sophisticated BIOS that has a swap table in it that you intend to change later using a utility, just make sure that the utility forces a relogging of the swapped drives (or all drives) after the swap has been implemented.

Listing 2. Modified select-disk BIOS routine that also swaps logical drives B and C.

```
ISELDK: LD      HL,(DRVEC)      ; Get drive vector
        LD      A,16            ; Get 16-drive into B
        SUB     C
        LD      B,A
ISELDSK1:
        ADD     HL,HL
        DJNZ    ISELDSK1
        LD      HL,0            ; Value for invalid drive
        RET     NC              ; Return if invalid drive

        LD      A,C             ; Get disk requested
        CP      2               ; See if it's C:
        JR      NZ,ISELDSK2     ; Skip if not
        LD      C,1             ; If so, change to B:
ISELDSK2:
        CP      1               ; See if it's B:
        JR      NZ,ISELDSK3     ; Skip if not
        LD      C,2             ; If so, change to C:
ISELDSK3:
        LD      A,27            ; Now log in drive
        JR      DOBIOS
```

Listing 3. Modified list output routine. It checks the printer width byte in the environment. If the value is 0, printer output is disabled by simply returning without calling the real BIOS list output routine.

```
PCOL    EQU     Z3ENV + 37H     ; Address of width data

ILIST:  LD      A,(PCOL)        ; See if printer width is
        OR      A               ; ..set to 0
        RET     Z               ; If so, just return
        LD      A,15            ; Else, call BIOS
        JR      DOBIOS
```

Disabling the LIST Device

I think it was our editor Chris McEwen who raised this issue with me. As I recall, he was unable to use a particular utility on his Z-Node because it had a function--not disabled when the wheel byte was off--that engaged the printer.

I had faced a similar problem myself. I have no printer attached to most of my computers, and occasionally I would accidentally hit a key that would initiate a printing operation. On some of the computers this meant instant crash! The BIOS would wait forever for the printer to signal that it was ready, and if I didn't want to wait that long, I had to hit the little red button, losing all my work.

My simple solution to this was to go into the BIOS and replace the jump instruction for the LIST function with a simple RET. Boy could that printer print fast! A little POKE instruction in my startup alias could handle this for me very nicely.

When Chris presented his problem, I told him he could use the BIOS in his NZCOM to take care of things. Just make up two versions of the BIOS, one normal version and one with the list routine RET'd out. Then just use NZCOM or JETLDR to install the one needed.

The code shown in Listing 3 is a more elegant solution. It looks at the printer configuration data stored in the environment. If the width is set to zero, then the print output is disabled. Otherwise it functions normally. As you see, the code is short and sweet.

Console Input/Output Enhancements

George Worley asked on Z-Node Central for a suggestion as to how he could get his system to send some escape sequences to his terminal whenever he pressed certain keys. Again, the NZCOM BIOS can solve the problem.

I originally wrote a virtual BIOS that would remap some keys on the keyboard. I make it interchange the 'a' and 'b' keys--not likely to be very useful, but it illustrated the point. I'm not going to show you that code; it is quite similar to the code for swapping drives except for one detail that will be covered in the example I will present.

The interesting thing about George's problem is that something going on in the console input routine is supposed to initiate an activity with the console output routine. The notion of mixing up the BIOS functions caught my interest. I decided to write a BIOS that would change the cursor on my Televideo terminal to a blinking block when I typed a tilde and back to a blinking underline when I typed a back apostrophe. See Listing 4 for the result.

The thing that is different here from the earlier examples with the SELDSK function is that in those cases the action was taken with input data, before the function was called. Here we must take action on data returned by the function, after the function has executed. Instead of jumping to DOBIOS, we call it and then continue with our code on return from it.

Once we have the character returned by CONIN, we check to see if it is either of our trigger characters. If not, we just return to the calling program with the character. If we do detect one of the trigger characters, then we send a string of characters to the screen using the CONOUT function. When that operation is complete, we then get the typed character back into its proper register and return.

I hope these examples will get you thinking about new ways to use the virtual BIOS. I have shown only very simple

code. NZCOM allows one to declare as much space as one wants for a virtual BIOS, and someday I would like to see someone write a version of BYE that can be loaded as a virtual BIOS.

Z-System for MS-DOS

I'd like to finish with a brief announcement. I had originally hoped to discuss this in more detail, but there just is not time or space, so I will leave it for the next issue. But I do want you to know about it now.

I'm sure I'm not the only Z-System user who also uses MS-DOS computers and finds DOS's primitiveness annoying and frustrating. Well, the new version 2 release of PCED (Professional Command line EDitor) is a DOS enhancement product that comes as close as any I have yet found to bringing the features we love in Z-System to MS-DOS. This is not entirely accidental, as I made the author aware of our Z-System work. As a result, PCED gives one most of the functionality of LSH and ARUNZ: full command history, both line and screen oriented, with editing and searching; multiple commands on a line; command scripts with advanced parameter parsing. There are some Z-System features that PCED does not add to DOS, but there are also many very powerful features it does bring that are probably only possible with the larger memory available on a DOS machine.

As is my wont with products like this that I use myself and really like, I got Sage Microsystems East to add it to the product line. PCED is now available at a very attractive price of only $50. I'll try to tell you more about it next time. ●

Listing 4. Modified CONIN routine. When particular characters are typed at the keyboard, escape sequences are sent to the screen. In this particular example, typing a tilde causes the escape sequence to select a blinking block cursor to be sent to the screen, while typing a back apostrophe sets the cursor to a blinking underline (for my Televideo terminal). Thanks to Howard Goldstein for this improvement to my original code.

```
ICONIN: LD      A,9             ; Perform the BIOS call
        CALL    DOBIOS
        CP      '~'             ; If tilde, send out
        JR      Z,SEQ1          ; ..sequence 1
        CP      '`'             ; If not back apostrophe,
        RET     NZ              ; ..return to calling
                                ; .. program
; Back apostrophe was typed
        PUSH    AF              ; Save input character
        LD      A,'3'
        JR      SEQ

SEQ1:   PUSH    AF              ; Save input character
        LD      A,'1'

SEQ:    LD      (POKE+1),A      ; Poke in final character
        PUSH    BC
        LD      C,1BH           ; Send escape to screen
        CALL    ICONOT
        LD      C,'.'           ; Send period to screen
        CALL    ICONOT
POKE:   LD      C,$-$           ; Filled in from above
        CALL    ICONOT          ; Send last char to screen
        POP     BC
        POP     AF              ; Get input character back
        RET
```

continued from page 37

guages, after all!

Get Your Feet Wet

No one ever learned how to swim without going into the water. Get a public domain program that includes source code. Use your assembler to reassemble/link it, and then satisfy yourself that the resulting COM file is the same as the original one. This is not an exercise in futility! It gets you familiar with the mechanics of your assembler and linker, so you can think more about programming and less about operating the assembler.

While you are experimenting with small programs, look forward to the next phase by reading the excellent articles by Bender (reference 4), and the tutorial by Meyer (reference 6). Your experimental program should be a standard .COM file, not something more exotic. Assemblers, linkers, and MLOAD (or the newer MYLOAD) are designed to produce .COM files when no other forms are specified. So your code will begin at address 0100H. The last instruction performed by your experimental program must return control to the operating system if you expect to continue use of your computer without resorting to a cold boot restart! The safe method is to make the final instruction a "JP 0000H", which causes a warm boot followed by return to the Command Processor.

If you need help, talk to your nearest AL programmer. Is there a subject you would like to see in more detail in the pages of *TCJ*? Tell the author about it, or write to the editor. [Ed Note: This is good material for the *Reader-to-Reader* column] The author can be reached via modem at (213) 670-9465, Z-Node #2 (Z-Node Central).

What ever happened to John Poplett? He soon quit technical writing and began programming professionally. He learned C, then learned assembly language for the VAX and for the 80x86 series of CPUs. The last time I heard from him he had tackled OCCAM, the assembly language for parallel processing transputers, and was busy with a C compiler for parallel processing.●

References:

(1) *Z80-CPU Technical Manual*, Zilog, Inc., Campbell, CA (408) 370-8016

(2) *Hitachi HD64180 8-bit High Integration CMOS Microprocessor Data Book*, Hitachi America, Ltd., San Jose, CA (800) 448-2244

(3) Bruce Morgen, "REL-Style Assembly Language for CP/M" *TCJ*, Number 35, Nov 1988, "Part 1: Choose Your Weapons" *TCJ*, Number 36, Jan 1989, "Part 2: Getting started"

(4) Andrew Bender, "Relocating Assemblers and Linkage Editors" *Microsystems*, Vol 4 No 9, Sept. 1983, page 86 *Microsystems*, Vol 4 No 10, Oct. 1983, page 114 *Microsystems*, Vol 5 No 1, Jan. 1984, page 120

(5) Dennis N. Quinn, "Structured Programming with M80" *Micro/Systems Journal*, Vol. 1 No. 3, Jul/Aug 1985, page 26

(6) Eric Meyer, *Introduction to Assembly Language Programming* (A 10 chapter tutorial available in disk file form under the name "MEYERTUT.LBR" available on Z-Node #2)

(7) Rodney Zaks, *Programming the Z80*, Sybex, 1979 ISBN 0-89588-013-X

(8) Kathe Spracklen, *Z-80 and 8080 Assembly Language Programming*, Hayden Book Co., Inc., 1979 ISBN 0-8104-5167-0 LCCC No. 79-65355

(9) William Wickes, *Logic Design With Integrated Circuits*, John Wiley & Sons, Inc, 1968 LCCC No. 68-21185

(10) *8080 Assembly Language Programmers Manual*, Intel Corp., Cupertino, CA. Intel ref. number 98-004C (1976)

PMATE / ZMATE Macros

2. Terminology and Utility Subroutines

By Clif Kinne

Notations for This Column

A Shorthand for Macro Names

In our first column we had two macros, which we named, 'D' and '^D', and referred to them, as such,- enclosed in single quotation marks. It bothers me that we also use quotes around single characters for purposes other than to signify a macro name. We could get around that by always saying, for example, "the macro 'D'". However, if we agree, we can adopt an even more concise and more nearly unique shorthand name, which is the macro call itself,- .D in our example.

The only conflict I can think of is that of the buffer calls, .0 through .9, with the corresponding decimal fractions. So, until I am persuaded otherwise, I shall use .D as shorthand for: "the macro whose name is 'D'". As a corollary, I propose to use .(n) as shorthand for: "the macro represented by the character whose ASCII number is n". This will allow us to identify macros where n = 128..255, as well as giving us alternatives for control characters and other awkward situations.

Radix Considerations in the Macro Listings

For ease of reading, any numbers in the source code listings for the macros will be in decimal,- base 10. When clarity is enhanced by using an ASCII representation of the number, that will be done; e.g., for 'insert a semicolon' ";I is clearer than 59I, which might require consulting an ASCII table. On the other hand, 75QX expresses what you are doing much more clearly than "KQX

However, if you ever expect to be working in hex (or octal?) as Jay does on occasion, he cannot urge you too strongly to use radix-invariant number representation. That means an expression in one or more ASCII characters for numbers greater than 9 (7 for octal). For example, to move down 150 lines, the following two command lines are equivalent:

```
(1)     150L            (with base 10 radix)
(2)     ''K*2L          (with any radix > 2)
```

This is no problem except for 13 and 32, the CR and SPACE characters. It is disturbing to have a macro insert a new line when you want it to say "^MQX, for example. To avoid that, I shall use 13QX in this column, but, for radix invariance, you should convert it to

```
(3)     ''M-''@QX,      for example, or
(4)     ''4/4QX         to save a byte.
```

In these columns I am also going to avoid the radix invariant form for the ASCII number of the SPACE character. In printed text it is hard to be sure whether a space is a space. I shall use 32 instead for that reason.

Incidentally, if you ever want to change all decimal numbers to hex, they can be found very easily using the .D macro presented in our first column. If you have .D in your permacs by now, get the permanent macros into an empty buffer with QMG, go to the top, and repeatedly execute

```
(5)     [.D@1>9]
```

It should stop at every number greater than 9: an unanticipated and unorthodox use for our first macro.

MATE vs. PCMATE vs. ZMATE

This is another area where we shall have to keep clear what we are talking about. First, I propose that we recognize the three categories, rather than just "8-bit" and "16-bit" systems. I suggest the above names as identifiers. I left the P off PMATE, to minimize confusion with PCMATE. We can then use PMATE as a generic term for all three.

The differences in macros written for the three systems are slight, and we hope it will rarely be necessary to list a macro for more than one. As long as we keep aware of the differences (see Table 1.) we should be all right.

I do not propose anything like an exhaustive discussion of these differences at this juncture, but let me cite a few consequences of these differences that bear on macros. As always, we shall appreciate your advising us of additions or corrections that should be made to this table.

1. Numeric variables.

With judicious use of the 100 variables available in PCMATE it should rarely, if ever, be necessary to save variables on the stack. If that turns out to be true, the extra byte entailed in setting and referencing variables 10-99 may be a small price to pay.

If you use the single-digit numbers, 0..9, in PCMate, you will have to be careful. If the following command starts with a digit, separate the two, preferably with an ESCAPE. Otherwise PCMATE will read them as a 2-digit variable.

Clif Kinne is a retired computer designer. He cut his teeth on vacuum tube and acoustic delay line machines in the fifties, made the transition to transistors and magnetic cores in the sixties, left the field to his children in the seventies, and tried, vainly, to catch back up with them in the eighties. He can be reached by voice at 617-444-9055, or via a message on Jay's BBS, 617-965-7259. His address is 159 Dedham Ave., Needham, MA 02192

2. Preloaded variables.

This has caused me some tergiversation. First I made the autoexec macro save @0, @1, and @2 in V90, V91, and V92, so I could save the extra byte most of the time. Then I found I was adding an escape, or a leading 0, to avoid the 2-digit syndrome. Furthermore, I was apprehensive that PCMATE might be doing something with V3..V9 sometime. So I switched to using V10..V99 exclusively.

Having done that, however, I find that I am disturbed by how much I lengthen a macro when I translate it from MATE to PCMATE, often by a dozen or two bytes. And I feel a lot easier about the risk of some other use of variables by PCMATE.

Now that I am writing this column, I am going to reverse myself again, except that I shall save @0, @1, & @2 on the stack instead of in V90, 91, & 92. That will help my macros to be usable in all three PMATES. I shall also find out something about how often I am inadvertently losing the stack by aborting out of macros.

Readers who only work in PCMATE should feel free to take advantage of the 90 extra variables.

3. Support of B@SE, B@3C, B@6M, et cetera

Fortunately, two of the three PMATEs support this. For one thing, this command makes it easier to use buffers without destroying their contents. We saw an example of that in the first of these columns last issue. For MATE, which does not support it, there are ways to guard against loss, and we shall offer macros to help in this. One, the Buffer Test, .^B, is included as one of the utility macros in Listing 1. I am going to interrupt this rambling here to discuss utility subroutines in general, and my own most used utilities in particular, as prelude to further remarks.

Macros for This Issue: Utility Subroutines

Macros for Single-character User Input.

See Listing 1

I believe it was a good three years before I gave any thought to utility subroutines. When I began to get "NOT ENOUGH ROOM IN PERMANENT MACRO AREA" fairly often, I began to look for ways to get more room without adding another allocation block to the size of MATE itself. So, I thought I would look through the permacs for repeated strings and see what could be done. Sure enough. About the worst repeater was the upper/lower case-independent test for the key struck in response to a command-line question:

```
(6)     @K=''S!(@K=''s)  (13 bytes)
```

where 'S' stands for any upper case letter. This can be shortened to

```
(7)     @K&95=''S       or its radix-invariant form:
(8)     @K&''_=''S      (8 bytes).
```

The drawback with these latter two is that they cannot be used if admissible answers to the questions asked include digits or other characters with ASCII numbers between 32 and 64. I had not given this much thought until I got PCMate early this year and found macros that came with it using the test:

```
(9)     @K!32=''s
```

Sure enough: 'ORing' a byte with 32 (0010 0000 bin.) sets bit 5, converts upper case letters to lower case, and moves control characters up by 32 ASCII. But the digits and others have bit 5 set already, so are unaffected.

On the other hand, ANDing a byte with 95 (0101 1111), as in (8) above, resets bit 5 (as well as bit 7), thereby corrupting the digits (and adjacent characters) to control characters. So I am converting to PCMate's form of the test, and am listing here subroutines based on that form.

Listing 1. Macros for single character user input.

```
^X^A    ;Answer.                                          15 bytes

;     FUNCTIONAL SPECIFICATION:   ORs the byte in @K with
;          bit 5, converting upper case responses to lower.
;          Compares this with the argument preceding the
;          call.  IF equal, returns with TRUE on the stack.
;          ELSE returns with FALSE on the stack.

;     USAGE:          Typical Call:   ''s.^A@S

;     LIMITATIONS:    The calling argument, ''s ,
;                     cannot be a control character.

@K!32   ;IF the character typed converted to lower case   1
=       ;is equal to                                      2
(@A!32) ;the calling argument converted to lower case,    3
  ,     ;  THEN push TRUE on the stack, ELSE push FALSE.  4

;Compact form:  @K!32=(@A!32),

^X^Y    ;Yes.                                             4 bytes

;     FUNCTIONAL SPECIFICATION: IF ''Y'' or ''y'' is typed,
;          returns with TRUE on the stack; ELSE with FALSE.

;     SUBROUTINES USED:  .^A, 'Answer' subroutine
;     USAGE:             Typical Call:   .^Y@S

''Y.^A;  Invoke the Answer macro with ''Y as the argument.

^X^C    ;Confirm                                          50 bytes

;     FUNCTIONAL SPECIFICATION:   Asks if okay to delete
;          or overwrite.  IF Y is typed, returns with TRUE
;          on the stack; ELSE with FALSE.

;     SUBROUTINES USED:  .^Y, 'Yes' subroutine
;     USAGE:             Typical Call:   .^C@S

GOkay to delete or overwrite?$  ;Ask the question.        1
.^Y             ;Invoke the .^Y macro.                    2

^X^B    ;Buffer test.                                     16 bytes

;     FUNCTIONAL SPECIFICATION;  Tests whether current
;          buffer is empty.  IF so, returns with TRUE on
;          the stack.  ELSE asks if okay to delete buffer.
;          IF Y, returns with TRUE on the stack;
;          ELSE  with FALSE.

;     SUBROUTINES USED:  .^C, 'Confirm' subroutine

;     USAGE:  When writing a macro which will delete or
;               overwrite a buffer, call .^B while in that
;               buffer to avoid unintended loss of its
;               contents.
;               .^B should shortly be tested by @S, with the
;                program continued or aborted accordingly.

;   Recall that:
;       @C = 0 at Top of Buffer
;       @T = 0 at End of Buffer.
;       @C!@T = 0 only if both @C and @T are 0, which is
;                       TRUE only for an empty buffer.
```

The Answer Subroutine, .^A

This is the basic subroutine of this group. The character to which @K is to be compared is passed to it as a leading numeric argument. It makes the case-independent comparison and sets the top of the stack TRUE or FALSE, accordingly. After a call the stack must be POPped (by @S or @S') to govern the desired action.

It could have been held to a length of 9 bytes, by simply changing "s to @A in (9) and adding a comma:

```
(9)     @K!32=''s       8 bytes
(10)    @K!32=@A,       9 bytes
```

However, by adding 5 bytes I can save myself the vexation of forgetting to make the calling argument lower case:

```
(11)    @K!32=(@A!32),  14 bytes
```

You can see from USAGE in Listing 1 that a call plus the POP take 6 bytes, so I am saving only two bytes per call over the 8 bytes in (8) or (9). However, the real savings come, not from shortening this macro, but from making it available as a permac callable by other subroutine macros with a passed byte argument.

Listing 2. Macros to expedite subroutine calls.

```
^X^G    ;GoBack (for MATE only)                         106
bytes

;       FUNCTIONAL SPECIFICATION:   Returns to the 'home'
;           buffer, which has been saved in V7 (by @BV7).

;       USAGE:  @7.^G

@A=@B%          ;IF still in home buffer, do nothing.     1
@A=0{BTE}       ;IF @B was 0, return to T Buffer.         2
@A=1{B0E}       ;IF @B was 1, return to Buffer 0.         3
@A=2{B1E}       ;IF @B was 2, return to Buffer 1.         4
@A=3{B2E}       ;IF @B was 3, return to Buffer 2.         5
@A=4{B3E}       ;IF @B was 4, return to Buffer 3.         6
@A=5{B4E}       ;IF @B was 5, return to Buffer 4.         7
@A=6{B5E}       ;IF @B was 6, return to Buffer 5.         8
@A=7{B6E}       ;IF @B was 7, return to Buffer 6.         9
@A=8{B7E}       ;IF @B was 8, return to Buffer 7.        10
@A=9{B8E}       ;IF @B was 9, return to Buffer 8.        11
@A=10{B9E}      ;IF @B was 10, return to Buffer 9.       12

^X^S    ;SaveEnv                                        27 bytes

;       FUNCTIONAL SPECIFICATION:   Pushes variables 0, 1,
;           2, 7, 8, and cursor position on the stack.
;           Loads variable, V7, with current buffer, @B.
;
;       USAGE:  Normally called at the start of any macro
;        which will alter more than one of the items saved.
;
@0,@1,@2,@7,@8, ;Push variables V0,V1,V2,V7,V8 on stack.  1
@X,@L,          ;Push current col. and line no. on stack. 2
@BV7            ;Put current buffer no. in variable, V7.  3

^X^R    ;Restore                                        34 bytes

;       FUNCTIONAL SPECIFICATION:   Restores the
;       environment saved by the SaveEnv macro.
;
;       STACK USE:      7 out of the 16 levels available.
;       USAGE:  Must be invoked when exiting a macro which
;               has used SaveEnv, .^S, whether that exit
;               was normal, a conditional exit, a jump, or
;               a programmed abort.
;               If the macro is aborted with  Ctrl-C or
;               terminated accidentally, the variables must
;               be reinitialized, manually or otherwise.

@7.^G           ;GoBack to 'home' buffer (MATE).  If you  1
                ;  are using ZMATE or PCMATE, replace this
                ;  with B@7E.
@S-@LL          ;Move from cur. line (@L) to saved line.  2
@SQX            ;Move to saved column no. (@S).           3
```

The Y(es), N(o), and Q(uit) Subroutines.

Next I found that, among macros usually in memory, I was comparing @K with "Y 7 times. with "N 6 times, and with "Q 4 times. Having the Answer subroutine already in memory, I could justify writing 3 new 4-byte subroutines:

```
        NAME            CODE        USAGE
    'Yes'  or  .^Y:     ''Y.^A      .^Y@S
    'No'   or  .^N:     ''N.^A      .^N@S
    'Quit' or  .^Q:     ''Q.^A      .^Q@S
```

Source listings for these are so trivial that I have included but one, .^Y, just for completeness.

You will have noticed that there is an extra flexibility in these subroutines that was not available with the raw testing of @K. The comparison can be made when convenient, but the result made use of later on in the program. Thus, for one thing, you could ask several questions, pile up the answers on the stack, then govern some action by a complex boolean expression through judicious use of the POPs. (e.g.: (@S&@S')!@S).

Next, if you look through your macros for occasions when you have compared @K with "Y or "N, you may be able to make up a subroutine that is a great space saver. I found that I was asking for confirmation many times before overwriting a file or deleting a buffer. This led to the next example:

The Confirm Subroutine, .^C

As you see from the listing, this routine is just as simple as .Y, having only two commands. Because of the string, though, it is 50 bytes long, compared to 4. Since it takes only 4 bytes to make use of it, it is a wonderful time and space saver in writing macros.

You note that .^C calls .^Y, which calls .^A. Let me finish

Listing 3. A subroutine to accept a string of characters.

```
^X^P    ;Prompt                                         68
bytes

;       FUNCTIONAL SPECIFICATION:  Inserts 3 blank lines and
;           displays, as a prompt, the one string argument
;           passed to it.  Tags the string response, which
;           is terminated with a CR.

;       VARIABLES USED:  V8,- flags calling macro if an ESC
;           was typed, so it will abort.

;       USAGE:  The caller will normally save V8 and the
;           cursor position on the stack before calling and
;           and restore them, as well as the CRT screen, at
;           the end.

0L              ;Move to column 0, current line.          1
3[13I]          ;Open up 3 lines to set off prompt.       2
-2L             ;Move to middle line                      3
9I              ;Tab, to indent prompt (for esthetics).   4
QAI^AA$         ;Insert the one string argument (prompt). 5
9I              ;Another Tab (more esthetics).            6
T               ;Tag beginning of user input.             7
[               ;Begin an iteration on keystrokes:        8
  GEsc to abort$; A comand-line message.                  9
  @K=27V8       ;  IF an ESC (27) was typed, store TRUE  10
;                 in V8, ELSE store FALSE.
  @K=13!@8_     ;  IF a CR or an ESC, escape loop.       11
  @K=127{       ;  IF a DELETE (127),                    12
    -D^}        ;    delete previous character and loop. 13
```

PMATE SYSTEM -->	MATE	ZMATE	PCMATE
Numeric variables	10 (1-digit)	10 (1-digit)	100 (2-digit)
Variables preloaded by PMATE. (if system has a built-in calendar/clock)	None	None	@0 = Day @1 = Month @2 = Year
Supports B@SE, etc.	No	Yes	Yes
Supports ..b (execution of buffer b beginning at the Cursor position).	No	No	Yes

Table 1. Differences among MATE, ZMATE, & PCMATE identified to date.

off this chain, for now, with a routine which calls .^C.

The Buffer Test Subroutine, .^B

I mentioned earlier that there are ways to guard against destroying your buffers, even with MATE. Well, this is one of them. If the buffer is empty, your macro can go right ahead and use it, without bothering you. If it is not, it will ask if okay to destroy it. If yes, it can go ahead. If not, your calling routine will have to decide what to do.

Macros to Expedite Subroutine Calls. See Listing 2.

The GoBack (to home buffer) subroutine, .^G

This subroutine enables MATE macros to return to a buffer previously saved on the stack or in a Variable. For ZMATE and PCMATE, B@SE performs that function much more concisely.

The SaveEnv and Restore Subroutines, .^S and .^R

These two save and restore the environment when your macro is going to move the cursor, go to a different buffer, or simply use certain variables. I must confess that I am just now adding these to my own permac area. What pushed me into it was the thought of having to write much of them over and over again,- with comments,- in each new macro listing.

Consequently, I expect to be adding to and modifying, them over the next several columns. In particular, I wonder if using up 7 stack levels will prove to excessively constrain any macros which would otherwise call .^R.

A Macro to Accept a Multi-character Response. See Listing 3.

The Prompt Macro, .^P

This macro is simply an attempt to extract all of the code common to several macros which ask the user for a multi-character response. It displays a prompt passed to it and tags the string response.

For a string of digits, the caller may store the number in a variable, with #.D (using our decimal macro from the first column). Other strings are usually moved to a buffer, with #B2C, for example.

This should be much clearer with an example, which [follows:][will have to wait for another issue.]

If computers ever become too powerful, we'll just organize them into a committee.

That'll do them in!

A Search Macro with Optional Change.

The Change Macro, .C

This macro invokes .^P twice,- first for the 'search' string, and then for the 'replace' string. It searches in a forward direction only and starts wherever you place the cursor. It is my feeling that options, such as 'search direction' and 'global', are more trouble to answer than it is to move the cursor to where you want to start (at least 99% of the time).

This is most useful, of course, when bound to a key so it can be called as an instant command. I have it invoked by ALT C, in both MATE and PCMATE.●

Listing 4. A search macro with optional change.

```
^XC     ;Change                                          97 bytes

;       FUNCTIONAL SPECIFICATION:  Calls .^P to display the
;          prompt, 'Change:'.  Waits for user's response,
;          terminated with a CR.  Moves response to Buf. 2.
;          Calls .^P to display To:'.  Moves s response to
;          Buf. 3.  Clears the prompt area and performs an
;           iterative search for buffer 2 contents with
;          optional change to buffer 3 contents.
;
;       VARIABLES USED: V8  Set by .^P if user types an ESC
;       BUFFERS USED:   2       Holds search string.
;                       3       Holds replace string.
;       SUBROUTINES:    .^S     SaveEnv.
;                       .^R     Restore.
;                       .^P     Prompt
;
;       SIDE EFFECTS:   As written, this will destroy any
;                      contents of buffers 2 and 3.
;
;       USAGE:  Before calling, the cursor should be moved
;        to or above the line where the search is to start.
;
.^S             ;Save the environment.                      1
.^PChange:$     ;Display request for string to change.      2
@8'             ;If an ESC was not typed by the user,       3
  {#B2C}        ;  Move the string typed to buffer 2.       4
.^PTo:$         ;Request replacement string from user.      5
@8'             ;If an abort (ESC) was not requested.       6
  {#B3C}        ;  Move the string typed to buffer 3.       7
-2L6K           ;Remove the entire prompt area.             8
[               ;Start loop.                                9
  @8_           ;  IF an abort was requested, escape loop10
  EUS^A@2$      ;  ELSE search for string to change.       11
  @E_           ;  IF not found, escape loop.              12
  GRET to change, ESC to terminate.$   ;Cmd line Msg.      13
  @K=13         ;  IF a CR was typed,                      14
  {             ;                                         15
    -C^A@2$^A@3$;    Make the change.                     16
  }             ;                                         17
  @K=27         ;  IF an ESC, escape loop.                 18
]               ;ELSE loop & continue search.             19
.^R             ;Restore the environment.                  20
```

Z-Best Software

Birth of a New Program

By Bill Tishey

Anyone who accessed the message board on Jay Sage's Z-Node this past December and followed the thread on development of XFOR.COM by Gene Pizzetta had a real treat. Not only did you get to follow development of a program from its conception to release, but you were able to witness the excitement and enthusiasm of user-involvement in its development.

Background: ZFILES.LST and the Z-SUS Database

This whole episode should probably start with a discussion of ZFILES.LST which I have been updating monthly for the past several years. ZFILES.LST was a list of Z-System program and utility versions (released and pending), originally published by Echelon in their Newsletter Z-NEWS #602 (10/6/86). The list at that time contained 130 programs, and only consisted of the filenames and version numbers for each program. Steven Gold updated this list in April, 1988, using the same format, and I began updating the file a few months later, but expanded the format to include, on a single line: the filename, latest version number, latest version of ZCPR supported, date issued, size in kilobytes, size in records, CRC, a remarks field, and, more recently, the Z-SUS distribution disk on which the program appears. In short, ZFILES.LST has been used to catalog the latest Z-System programs and utilities and to provide some essential statistics for each pertinent .COM file. Figure 1 shows the header and a typical entry. Note that an asterick ("*") preceding the "Remarks" field indicates a commercial or proprietary program and "?" indicates a pending program (under beta-testing).

```
                S
                Y
Name      Vers  S ZSUS Issue Size Recs CRC  Remarks
========= ====  = ==== ===== ==== = === ===  ================================
FOR       1.20  0 V203 01/91   4k   29 7715  XFOR12.COM
```

Figure 1

ZFILES has continued to follow this format. This past October, however, after several weeks of rooting through my archives (which represent months of download time!) and reorganizing and updating data, I put together a fairly comprehensive database of the Z-System files released over the major Z-Nodes during the past five years. Though still not complete, I think it's a good beginning toward cataloging not only all the great programs and utilities, but also the aliases, patches, tips, et cetera, which continue to be generated in support of Z-System. The database is being maintained in dBase III+ on my '286 clone. As of this writing, it contains over 700 entries. I've been transferring various pulls from the database to my CP/M-equipped (PCPI CP/M, NZCOM, ZSDOS) Apple II+ via a null modem and doing final-editing in WordStar 4.0.

```
                S
                Y
Name          Vers   S ZSUS Siz Rec CRC  Library/Size Issued   Author
============ ====   = ==== === === ===  ============ ======== ================
XFOR          1.20   0 V203   4  29 7715 XFOR12     38 12/23/90 Gene Pizzetta
   Z-System FOR utility for displaying ''----'' delimited file catalogs.
   Command line source file specification. Numerous configuration options
   can be set with ZCNFG.
```

Figure 2

The first product of the Z-SUS database was ZFILEV01.LST, a "verbose" version of ZFILESxx.LST (the original list, now the "brief" version, has been renamed ZFILEBxx.LST and continues to be updated monthly). ZFILEV01.LST was in response to a number of requests to include a brief description of each of the program entries in ZFILES. Now, those who have no idea of what things like LSH, SNAP, TCSRC, OE, et cetera, are, can get a reasonable idea of what those programs do. ZFILEVxx.LST will be updated "periodically" (possibly monthly, but at

---

Bill Tishey has been a ZCPR user since 1985, when he found the right combination of ZCPR2 and Microsoft's Softcard CP/M for his three-year-old Apple II+. After graduating to ZCPR30 and PCPI's Applicard CP/M, he did a "manual install" of ZCPR3.3 (with help from a lot of friends!), and in late 1988 switched to NZCOM and ZSDOS, all on the same vintage Apple II+. Bill is the author of the Z3HELP system, a monthly-updated system of help files for Z-System programs, as well as comprehensive listings of available Z-System software. Bill is the editor of the Z-System Software Update Service and has compiled such offerings as the Z3COM package and the Z-System Programmer's Toolkit. Bill is a language analyst for the federal government and frequents the Foreign Language Forum (FLEFO) on Compuserve. He can be reached there (76320,22), on Genie (WATISHE), on Jay Sage's Z-Node #3 (617-965-7259) and by regular mail at 8335 Dubbs Drive, Severn, MD 21144.

least two or three times a year). Its header and a typical entry are shown in Figure 2.

Another purpose of ZFILEVxx.LST was to provide the name of the original library in which a Z program was distributed. Since the .COM files often have different names than the libraries in which they are distributed, this is very important to users attempting to track down the programs. The Z-System Software Update Service had also talked about offering to compile custom disks for its customers. Providing the size of the libraries made it easy for users to mix and match programs to fill a particular disk format.

The database also allows for grouping files and programs according to categories. While I've always maintained special disks for word processing, programming, communications, system, file, disk, print functions, et cetera, all files can now be pulled together according to these and other categories. Again, this has potential for use in compiling special packages for Z-SUS customers and has already helped significantly in producing the Z-System Programmer's Pack (see announcements in this issue).

The database is also helping me to keep track of programs for which .HLP files for my Z3HELP system (a subject for another column!) have yet to be written. I'm a firm believer in documentation. Taking the time to define the syntax, function, and usage, as well as the traps and limitations of a program, makes it much more meaningful and useful to those who would use it. For those who maintain a system with many types of programs, HELP files are a good way to organize and make this information easily accessible.

Other ideas I've had for "lists" from the database include:

1) programs supporting the various versions of ZCPR
2) programs supporting CP/M+
3) programs supporting the various types of datestamping (DateStamper/ZSDOS/Z80DOS)
4) programs for which .CFG files have/have not been written

Special packages can be compiled for: word processing, system control, file manipulation, datestamping, alias applications, et cetera. Maybe the readers have other ideas. If so, I'd like to hear of them. Now, let's get back to the XFOR saga.

A "Spruced-Up" FOR

When I released ZFILEVxx.LST this past November, Bob Dean made an excellent suggestion to make the list "FOR"-compatible by adding a "----" delimiter between each file record. FOR.COM, in its various adaptations (FORZ, ZFORP, etc.), is a tool used on remote access computer systems (RAS), originally created by Irv Hoff, to provide on-line descriptions of available files. FOR is configured to read one or more .FOR files which contain the actual descriptions and often reside in a private, protected area. Making ZFILEV "FOR"-compatible would allow users to scan ZFILEV online to view the essential stats and descriptions of particular programs in which they were interested.

Shortly after I released the modified ZFILEV, Chris McEwen noted that he had patched his existing FOR utility as well to read the file and was calling the resulting file DF (Describe File). Users on his RAS now could also scan the list for any number of things: function, program, author's name, date, etc. Chris was quick to notice that a modified FOR might be useful for both RAS and personal use in scanning all kinds of lists (including Ian Cotrill's listing of Remote RCP/M systems, RCPMmmyy.LST, which he updates monthly). He presented to Gene Pizzetta the idea of "sprucing up" FOR, and Gene took up the challenge. Almost immediately, Gene was deluged with "suggestions." He showed a great deal of patience and understanding, however, in handling suggestions from many sides and credit is due him for keeping everyone's interests at heart. The result is truly a program of great utility for everyone. I captured most of the discussion thread and offer below a fairly accurate chronology of events. What I've tried to show is how the comments and suggestions (left column) greatly influenced the development decisions (right column). Note that the comments have been heavily edited to save space. They were actually offered with much friendly discussion and not in the rather curt way they may appear.

Vs 0.1 (Dec 2, 1990)

Armed with the suggestions below, Gene set to work and, within a few days (!) released version 0.1. Everyone agreed that it was a good start. ZFOR, as it was initially named, was based on Carson Wilson's FORZ 1.0 (8/5/87), which was a revised disassembly of Irv Hoff's FOR.COM. As a standard Z tool, it responded to the "//" option on the command line, set the error flag on a non-match, sported a quiet mode, and could be re-invoked with the GO command. It was also configurable with ZCNFG.

Comments and Suggestions:	***Development:***
Consider the following syntax: FOR [for-file-name [search string]] By allowing the text file to be specified on the command line, a single FOR.COM could be used (via ARUNZ or other aliases) to work with various text files; one would not, as one does at present, need a separate COM file patched for each text file.	ZFOR adds command-line source file specification.
The way to handle the problem of looking at files that are not in the public area (some RAS's keep FOR files in a private area) is to provide an internal default path (settable with ZCNFG). If no explicit DU: is given, the file would be fetched from the default directory.	If no DU or DIR spec is given, an internally configured default directory is used or, if not configured, the currently logged directory.
Perhaps we could have a default filename and DU: for stand-alone use (those not using it in conjunction with ARUNZ or aliases).	
Yes, allow a default name and directory, but use that file only if the program is invoked with an empty command line tail. Otherwise require that the first token be the file spec and any remaining tokens be search words.	If no command tail is given, ZFOR defaults to the configured internal filename.

Vs 0.2 (Dec 8, 1990)

Some confusion resulted from users' misunderstanding of the command line parsing. Many configured an internal filename and DU and, using it in a stand-alone situation (not called from an alias), discovered they couldn't look for a search string without giving the filename. The result was some thinking that the new FOR didn't provide string-searching! Gene reassured everyone that this function had been in FOR from the beginning and that he had only made some very minor changes to it. Besides, an alias such as "FOR zfor a15:for $*" would allow you to simply type ">FOR string" to get the job done. Some, however, continued to insist on the flexibility for stand-alone use.

With this version, Gene added a three-way screen paging option: paging, continuous scroll (using ^S to pause), or asking the user whether to page or not. Leading spaces were also made significant. Only one space after the filespec is skipped; multiple spaces are part of the first search string. This allows looking for " ren" and not getting "referRENce".

Comments and Suggestions:

FOR is useless without string search, and ZFOR seems to accept a filename parameter OR a string. It needs both.

What would you think of the following syntax (and new name): "XFOR string"--use internal filename and directory; "XFOR [D][U]:file string"--use specified file. The colon indicates that the parameter is a filename.

Another name should be used rather than ZFOR as that name is used in ZMD.

You might delete the "----" between entries and highlight the entry as well. That might allow more data on the screen while still keeping it separate.

The entries should be separated either by the "----" or a CR/LF. To remove both is too much. For ZFILES, the display is a bit too cluttered.

I like the ZFILES header and would like to have a different one for the Nolan format (which includes the DU:, file size and date on the same line as the filename).

I kept getting the message that the buffer was overflowing, which ended the search.

Consider adding some additional abort characters more in line with Irv Hoff's FOR, specifically K,k (non-control).

How about the ability to use file wildcarding in the search string?

It would be nice to allow for multiple search words.

Development:

The syntax was changed so that, if the first token contains a colon, it is a filename. If either the DU/DIR spec or filename is missing, they are filled in from the configured defaults. This allows a string search to be made if a DU is given without a filename.

The name was changed to XFOR to avoid conflict with ZMD utilities.

The first line of each entry is highlighted if available from TCAP. Reverse video is used for header line, if available.

Configurable to allow a blank line between entries on screen.

Provides for an additional roll-your own header in the form of a patch file which can be overlaid into the program.

The entry buffer now handles single entries up to 8k.

XFOR can be aborted at any time with ^C, ^K, ^X, C, K, or X.

Search strings may include the following special characters: "|" - separates multiple search strings, "\" - matches the beginning of a line, "?" - matches any single character, allowing primitive wildcard searches.

Vs 1.0 (15 Dec, 1990)

In this, the distribution version, Gene changed the method of making leading spaces significant after discovering that the syntax would not work from an alias (ARUNZ purges the leading spaces). Leading spaces are now ignored unless the match string begins with " |". "XFOR A1: | REN", thus, searches an internally configured file in A1 for an entry beginning with "REN".

Gene also added use of direct cursor addressing and clear-to-end-of-string if available and if a header is displayed. This allows the header to be printed once and not flicker at the top of the screen.

Comments and Suggestions:

Suggest that TCAP use be optional and that the screen NOT be cleared (or at least be configurable).

Under the old TCAP, after the first few screen fulls of a ZFILES search, the first line in each file description has the spacing to the end of the line in reverse video.

Development:

Variable screen overlap and clearing the screen before each page are now configurable options.

Fixed problem causing highlighting not to be terminated on some machines.

Vs 1.1 (18 Dec, 1990)

Some minor changes were made in this version, including addition of a CR, LF after the paging question and the use of BOUT instead of COUT in the print loop so that tabs are expanded.

Comments and Suggestions:

It'd be nice to redirect output to a file, printer, or append it (ala CONCAT) to an existing file.

You might wish to add a way to turn on and off the highlighting of the first line of each entry. Perhaps ability to choose no reverse can be built into the ZCNFG file.

Development:

Added printer output and "+" command line option (last token preceded by a space).

Made video attributes used for header and first line of entry configurable options.

Vs 1.2 (23 Dec, 1990)

Just when everyone seemed adjusted to the new command line syntax, a suggestion was made to change it (where have I heard that before?): the creation of an option field preceded by a slash. This idea was too good to ignore, since it would simplify adding new options (as long as there was a good tokenizing routine) and avoid seemingly "glued-on" enhancements such as the "+" command for printer output.

Comments and Suggestions:

How about the following syntax: XFOR [[DIR]:[FILE] [/options] [strings] Some possible options include: H-header, P-page, S-space between lines, L-print, N-no paging, etc.

Development:

Changed command-line syntax: an option list, preceded by a slash can now be included in the command line, just before the match string. Options: H-display header, A-use alternate header, S-double space between entries, L-echo to printer, V-turn off all screen attributes, P-use screen paging, N-no screen paging.

Vs 1.3?

Finally, there were some lingering suggestions to consider for future updates. Some of these were dismissed for obvious reasons. The ability to read "crunched" file lists, for instance, had been suggested early on, since many of the file lists take up much disk space (ZFILEVxx.LST is 120+k). The speed tradeoff in uncruching, however, may not be worth adding this option.

The ability to expand an '*' to '???' in a search was also suggested but considered undesirable. Every special character added to the search mode, of course, is another character which cannot be searched for, and these should be kept to an absolute minimum. In addition, the XFOR search is looking for strings, not filenames. Entering "xf*" would have exactly the same effect as entering just "xf".

Comments and Suggestions:

Consider using a CR to advance a screen, a SP to advance a line only and adding the ,. standard command. Consider also backward scroll (like found in V.COM), if supported by extended TCAP.

Could you add "N" as an abort option at the end of each page and change the prompt from "More" to "More?". This would accommodate users not familiar with the RAS conventions (^C, ^K, etc.).

How about a new command ("S") while waiting for "More" to bring up a prompt looking for a new search string?

Development:

I hope this little, unfolding drama has not only informed you about XFOR, but also given you a taste of program development in the public domain (at least as it thrives among the Z-Nodes). As Bob Dean at one point noted: "One thing about CP/M compatible public domain, it always has a genealogy a mile long." Indeed, XFOR's parentage can now be traced from Gene Pizzetta (XFOR), to Carson Wilson (FORZ), to Irv Hoff (FOR), who evidently got the idea from CompuServe. In addition, the DU:, file size, and date fields were added to the FOR-file format by Gene Nolan and Bob Dean. As with most other CP/M and Z programs, credit is also due the many users and testers who took time to offer their suggestions and feedback. For XFOR, the major contributors were: Chris McEwen, Jay Sage, Bob Dean, Howard Schwartz, and yours truly.●

---

continued from page 18

room temperature is within some hysteresis band of the desired temperature, the circulator is cycled on for only one minute out of ten. Should the room temperature fall below the bottom limit point, then the circulator runs continuously. This approach has been very effective.

Design Goals

The above description should give you a pretty clear picture of the kind of sophisticated control that a microprocessor based system can provide. I would like to close this installment by saying a few things about some important design constraints I imposed.

I required that the system operate for an extended period of time without AC power. One would not want a power glitch to erase the programmed schedule and leave the electrical circuits and heating system uncontrolled thereafter. We might, after all, be away on vacation. A sealed lead-acid battery, constantly charged from the AC, can operate the complete system for at least 10 hours.

During the time that the power is off, some control operations cannot be performed because there is no electrical power to the circuits being controlled. The system is smart enough to know that AC power has failed, to keep track of any state changes that should have been carried out, and to carry them out as soon as power is restored.

A second requirement was that the system be highly failsafe. A complete failure of the computer must still result in a house whose temperature will stay within reasonable bounds. Failure of the controller with the boiler turned on must not lead to overheating and possible explosion of the boiler. Failure of the controller at a time when the boiler is turned off must not allow the house temperature to drop to the point where water pipes might freeze. We will leave the description of how this is done for next time.

Finally, there must be full manual backup control. Electrical circuits must be switchable as usual with the computer controller turned off or completely removed. Under the same circumstances, the heating system must fall back to normal manual control. This, too, will be described next time.●

continued from page 20

and addresses are still entered as hex quantities. In a third level of abstraction, numeric quantities and addresses are allowed to be represented symbolically, and the assembler takes over the chore of keeping track of the actual addresses. Instead of getting buried in the details of assembler operations, let's start over from the viewpoint of the high level language, the next level of abstraction.

From High Level to Assembler

High level languages deal with data structures built up from numbers and strings. Operations performed on such data structures include arithmetic, logical, and I/O operations. To be sure, integers may be defined as a byte and long integers as a word, but the emphasis is on the data type. CPU registers appear in C, where some compilers allow specification of an integer as a register variable. But the programmer does not specify which register is to be used; the compiler makes that choice. By contrast, assemblers deal *only* with registers and memory locations. Treatment of data as one type or another is entirely the choice of the programmer since a data structure is ultimately defined in terms of the operations which may be sensibly performed on the collection of data elements.

Assembly Language Programming: Mnemonics and Opcodes

As in any new language, you must become familiar with the actual instruction set. There are three commonly used instruction sets, Intel, Zilog, and Intel modified for Z80. The instruction sets are implemented by assemblers. For example, consider the instruction to copy the contents of the C register into the B register. The Intel mnemonic is "MOV B,C", the Zilog mnemonic is "LD B,C". Both result in the same byte of code processed by the cpu: 41H. M80 is able to translate either Intel or Zilog mnemonics. Other assemblers use only one set. Learn the set that your assembler uses by any means possible. The 8080 has 244 instructions; the Z80 has about 3 times that number. Don't despair, however. Most of the instructions are variations on a basic set of about 20 (depending on how you count differences). This part of assembly language is like learning your addition tables - you just have to do it. As you write programs, many of these instructions and their effects will become second nature. I don't remember the details of all of them, even after many years of AL programming. But I know they exist and look them up when necessary. If you own ZMAC, then you know that there is a companion HELP file named Z80.HLP. Written by Cam Cotrill, Z80.HLP provides all the details for every instruction used by the Z80 and Z180/HD64180 in quickly accessible form. The same data is to be found in tables included in references 1 and 2.

Assembler Instructions -- Pseudo-Ops

The cpu instructions discussed above are also referred to as *opcodes*, and *mnemonics*. They are assembler instructions, because they instruct the assembler to generate code for execution by the cpu. There is another class of assembler instructions, commonly called *pseudo-ops*. This second class does *not* directly produce executable code. An important pseudo-op is the DEFB statement, which defines the contents of a memory location. Another is the ORG statement, which defines the address at which subsequent code is to reside during execution. Other pseudo-ops provide for conditional assembly of code or other aspects of the operation of the assembler itself. The MACRO pseudo-op provides a powerful facility for HLL-like features; RMAC generates Z80 code with the help of Z80.LIB, a library of macros.

Pseudo-ops have always been a part of non-trivial assemblers. Each assembler, however, has seen fit to give different names to the same function! The DEFB function in M80 was named DB in ASM, MAC, and RMAC. Another synonym is DEFM. SLR assemblers recognize the synonyms that were being used by most assemblers of its day. ZMAC also recognizes and properly interprets pseudo-op synonyms, including those introduced by SLR. You will become acquainted with the set of pseudo-ops that your assembler uses. Knowing about synonyms becomes important when you attempt to read source code from someone else whose assembler is different.

Experienced AL programmers have become accustomed to the differences in source code languages, and work comfortably with all the variations. Not much different than High Level Lan-

continued page 28

8031 µController Modules

NEW!!!

Control–R II

√ Industry Standard 8–bit 8031 CPU
√ 128 bytes RAM / 8 K of EPROM
√ Socket for 8 Kbytes of Static RAM
√ 11.0592 MHz Operation
√ 14/16 bits of parallel I/O plus access to address, data and control signals on standard headers.
√ MAX232 Serial I/O (optional)
√ +5 volt single supply operation
√ Compact 3.50" x 4.5" size
√ Assembled & Tested, not a kit

$64.95 each

Control–R I

√ Industry Standard 8–bit 8031 CPU
√ 128 bytes RAM / 8K EPROM
√ 11.0592 MHz Operation
√ 14/16 bits of parallel I/O
√ MAX232 Serial I/O (optional)
√ +5 volt single supply operation
√ Compact 2.75" x 4.00" size
√ Assembled & Tested, not a kit

$39.95 each

Options:
• MAX232 I.C. ($6.95ea.)
• 6264 8K SRAM ($10.00ea.)

Development Software:
• PseudoSam 51 Software ($50.00) Level II MSDOS cross–assembler. Assemble 8031 code with a PC.

• PseudoMax 51 Software ($100.00) MSDOS cross–simulator. Test and debug 8031 code on your PC!

Ordering Information:
Check or Money Orders accepted. All orders add $3.00 S&H in Continental US or $6.00 for Alaska, Hawaii and Canada. Illinois residents must add 6.25% tax.

Cottage Resources Corporation
Suite 3–672, 1405 Stevenson Drive
Springfield, Illinois 62703
(217) 529-7679

continued from page 16

```
ACTION ,X LDX      ( get its action )
0 ,X JSR               ( fire it! )
```

CURRENT_MCB is a pointer to the currently active MCB. The linked list is used from within an action to propel the interrupt through the list, firing each MCB's action. The constraint on this is that all MCB actions MUST fire before the next interrupt hits. These states or routines must be methodically factored. If this system was to have 11 motors, the action and sensor routines would have to be very short! The list is completed when the Anchor MCB's action fires its RTI instruction. Once again, the list is a circularly linked list. The list is closed--the Anchor MCB points to the first MCB--the Last MCB points to the Anchor MCB.

```
INITIALIZE_MOTOR

CODE ADD_MOTOR_TO_LIST ( mcb -- mcb ; uses X, D ;; Adds a new MCB to the )
                       (            ; List of MCBs. )
ASSEMBLER
   ANCHOR_MCB # LDX
   NEXT_MOTOR ,X LDD
   0 ,Y LDX
   NEXT_MOTOR ,X STD

   0 ,Y LDD
   ANCHOR_MCB # LDX
   NEXT_MOTOR ,X STD

   NEXT ^ JMP
END-CODE

CODE FILL_MCB ( mcb action sensor timer port # -- mcb )
              (       ; uses X, D                    )
              (       ; Sets parameters in a new MCB  )
ASSEMBLER
    0A ,Y LDX
    01 ,Y A LDA  STEP_BIT ,X A STA  INY INY  ( Parameters are accessed  )
                                             ( from the data stack:     )
                                             ( <number> ,Y              )
                                             ( set to the appropriate   )
                                             ( variable,  then the      )
                                             ( element is poped off of  )
                                             ( the stack. )
                                             ( In Max-Forth 1 ,Y is the )
                                             ( LSB and 0 ,Y is the MSB. )
                                             ( The Stack Pointer is     )
                                             ( incremented .. INY ..    )
                                             ( last to remove an element)
                                             ( and decremented first to )
                                             ( add an element.          )
    00 ,Y LDD    >>PORT ,X STD      INY INY
    00 ,Y LDD    TIMER ,X STD       INY INY
    00 ,Y LDD    SENSOR ,X STD      INY INY
    00 ,Y LDD    ACTION ,X STD      INY INY
    NEXT ^ JMP
END-CODE

: ADD_MOTOR ( mcb action sensor timer port # -- )
   FILL_MCB   ADD_MOTOR_TO_LIST DROP ;

FIRST_MCB           ( MCB )
>_WAIT @            ( Action Routine )
>SENSOR_ACTION @    ( Sensor Routine )
DELAY_TIME @        ( Reload Timer count )
8000                ( Address of Motor Port )
01                  ( Address in Port of Motor )
ADD_MOTOR

: LEFT  ( -- ; motor left )
   DI >PORT C@
   FIRST_MCB STEP_BIT + C@
   OR
   >PORT C! EI
;

: RIGHT ( -- ; motor right )
   DI >PORT C@
   FIRST_MCB STEP_BIT + C@
   0FF XOR  AND
   >PORT C! EI
;


: RUN_MOTOR_1 ( -- ; spin it! )
   OC1F TMSK1 C!
   EI
```

Nuts and Bolts (And Tools!)

The code in listing 1. is the start up code. It is loaded into the NMIS-0021 first. The postfix assembler is loaded next. Finally the code in listing 2. is loaded into the single board. The following is a review of the highlites found in listing 2.

The motor interrupt handler uses the 68HC11's Output Compare Timer 1 (TOC1).The interrupt handler first acknowledges the interrupt. This prevents the firing of this interrupt upon interrupt exit. If the interrupt is not acknowledged, it will cause a recurring interrupt. Next the current 68HC11 clock count is read from TCNT, added with the TIMER_OFFSET variable, then stuck into the TOC1 register. TIMER_OFFSET in relation with DELAY_TIME determine the step settle time--TIMER_OFFSET is the fine tuning variable while DELAY_TIME is the coarse tuning variable. When TCNT catches up to the value in TOC1 another interrupt will fire. The first MCB is accessed through the Anchor MCB and the cycle continues.

In order to have things run smoothly, "things" must be set up properly.INITIALIZE_MOTOR connects the Anchor MCB to itself, sets the Anchor MCB's action to the Last action--the action that ends in an RTI. Finally the Interrupt vector for TOC1 is set in EEPROM.

The last words in listing 2. are tools used to construct a new MCB and connect it into the list. Note how FILL_MCB uses the Forth stack to get its parameters.

"Spin It On!"

The NMIS 7040 board is described quite well in New Micros documentation. I still referred to the Signetics linear manual for the actual connection of the SAA1027 chip to a motor. The SAA1027 is the heart of the '7040 board which has four SAA1027's built in.

continued page 39

continued from page 6

and switches, or provides minimal noise filtering that will be ineffectual in the face of an actual surge. Many users would be as well served with a $3 hardware store MOV protector that they discard and replace periodically, as they would with an expensive protector using the MOVs, which will also wear out.

Computer Reliability

As computers spread into more and more critical areas of industry and government, their reliability assumes greater importance. No longer is computer failure, especially in networks, just a matter of running down to the computer store for a new power supply or motherboard. In sophisticated computer installations, many people may be unable to do their normal work until their computers are restored. As Infonetics' 1989 study of network failure costs showed, total failure costs far exceed hardware expenses to repair equipment, although these extra costs are often considered uncontrollable and buried in other overhead expense.

In addition to physical damage, the potential for costly data errors and "no problem found" disruptions from powerline disturbances makes computer power protection a poor candidate for minor cost savings.

What Protection Do Computers Need?

Computers need powerline protection which does the following:

- Provides low let-through voltage (under 250 volts peak is harmless).
- Does not use the safety ground as a surge sink and preserves it for its role as voltage reference.
- Attenuates the fast rise times of all surges, to avoid stray coupling into computer circuitry.
- Intercepts all surge frequencies, including the high frequency internally generated surges.
- Does not convert normal mode surges into common mode.
- Does not degrade in service, or if it does, at least is thoroughly safe in the event of thermal runaway and employs reliable status indicator circuitry.

The Ideal Surge Protector

The ideal surge protector would disconnect the load from the powerline for the duration of the surge, then reconnect it. Since this is not possible with today's switch technology, a practical approach is to present a high impedance to the surge and a low impedance to the power wave, coupled with surge storage and frequency attenuation circuitry that will remove the disruption and damage potential from the surge, without using the critical reference ground as a surge sink, but rather only neutral as the return circuit.

Solution

Power protection is important to the reliable operation of computer networks. Once the critical role of the reference ground is understood, and how it provides a "back door" entry into the low-voltage logic circuitry of a computer, network managers can make informed decisions of how to protect their equipment. Protection which meets the criteria outlined above without diverting surges to the reference ground will reliably protect networks, while surge suppressers which use the ground as a surge sink may well exacerbate computer problems. Computer power supplies are likely to be more surge tolerant than low voltage logic circuitry, and the decision for network configurators should perhaps be reliable protection or no protection, but not the risky middle course of ground disrupting ordinary shunt suppressers.●

References

[1]"Changes Considered for UL 1449", *LAN TIMES*, July 1990, p.102.

[2]Martzloff, Francois D., "Coupling, Propagation, and Side Effects of Surges in an Industrial Building Wiring System", Conference Record of the IEEE-IAS 1988 Annual Meeting, pp.1467-1475.

[3]"Beware Those Network Failures", *The New York Times*, September 17, 1989.

[4]"Surge Protection Revisited", *LAN TIMES*, May 1990, p.89.

[5]"Power Line Protection--A Danger to Network Datalines", *Power Quality*, Premier Issue, 1990, p.104.

[6]"Transformer Parasitic Capacitance Affects Switcher Design", *PCIM*, May 1990, p.56.

[7]"Surge 'Protectors'--Worse Than Useless?", *Princeton MacIntosh Users' Group Newsletter*, June 1990.

[8]"Zero Surge Model ZS1800 Surge Eliminator--Product Review and Report", Vernon L. Chi, Department of Computer Science, UNC Chapel Hill, July 1990.

[9]"Adapting Adjustable Speed Drives to the Electrical Environment", *Power Quality*, Premier Issue, 1990, p.34

[10]Martzloff, Francois D., and Leedy, T. F., "Selecting Varistor Clamping Voltage: Lower Is Not Better!", Zurich EMC Symposium, April 1989.

[11]SAFE-ALERT Report of Fire Incident with MOV Surge Protector, Government-Industry Data Exchange Program, September 1988.

---

continued from page 38

This chip does the step commutation logic and drives the motor. My motor was a disk drive stepper motor, used to move the head to specific track locations. The NMIS 7040 board was jumpered to address 8000 hex on the 68HC11's address bus.

The motor control driver is built with the ADD_MOTOR word. All is pretty explanatory: the 8000 is the address in hex of the NMIS-7040 board, 01 is the actual motor address in the NMIS-7040 board. RUN_MOTOR_1 will start the motor spinning. LEFT will spin it left, RIGHT will spin it right. LEFT and RIGHT can be typed while the motor is spinning. By modifying the DELAY_TIME and TIMER_OFFSET variables while the motor is spinning, you can change the speed at which the motor spins. For now, use DI to stop the motor from spinning. Later we will see a more precise way of stopping the motor.

Until Next Time

A lot of information has been presented for one article. Read through the code--the fundamental paradigm found within can be applied to a plethora of applications in the embedded controls field. In the next Article we will build onto the MCB and control the motor's behavior in terms of its acceleration. If I can find a suitable and inexpensive encoder, we will incorporate that into our system. Until then have fun with Forth.●

continued from page 2

Control Yourself!

I was chatting with Jay Sage one evening last month. The talk was on embedded controllers. I wondered if there wasn't a field of interest for CP/M users here. Actually, this was a loaded question. I knew that Jay had built a complete home control system out of an 8085 box some years ago. The good news is that he took the bait. His first article describing this system is in this issue. Meanwhile, I baited Rick Swenton in Connecticut on X-10 modules. Not everyone is up to wiring their house to the extent that Jay has. As it happens, there is renewed message traffic on some of the systems around the country about X-10. Think we should go for it?

Of course, the real action in embedded controllers is in industrial applications. Frank Sergeant, the winner of the Harris competition, debuts in this issue with his project. It is a dedicated floppy disk alignment machine sufficiently small to allow alignment of drives in the field. No more oscilloscopes, no more downtime while the drive goes to the shop. Really a great article. Of course, it is not a trivial project and we will be presenting it to you in a couple of installments. Simply didn't have space to fit it all in one issue. This article shows the power of the Harris RTX chip and Forth in a real-world application.

Readers will remember Art saying that he wanted to have more time to spend on his own projects. That was one of the reasons he decided to retire as the publisher of *The Computer Journal.* Lucky Art! Zilog called to tell us of the new member of the Z80 family. The Z181 seems to be one hot chip--built in USART, timers, the works. Since this is right up Art's alley, I passed them on to him. Zilog sent him an application board and he is busy in the workshop putting the new chip through its paces. We will have his report later in the summer. Initial reaction: great! Be on the lookout for more on this as it develops.

Meanwhile, Matt Mercaldo continues with his series on stepper control in this issue. Pay close attention to this series of articles. Matt is leading us into robotics. In reading his "author's bio," I fully expect to see "six legged mechanical men" running around this place some time in late summer. Fascinating. My wife, Ester, says she can't wait. She has already arranged lodging with her sister.

As the Z-World Turns

Of course, *TCJ* wouldn't be *TCJ* without great articles on Z-System. Jay tackles an excellent topic this time around: modifying the NZCOM virtual BIOS. He puts out a general challenge for more work in this area. Applications are nearly endless, and whatever you can't accomplish in the BIOS, you can tackle with an IOP. Lindsay Haisley of Austin, Texas is hot of the trail with an upcoming article on that.

The real strength of the Z-System community is the way people work together. Bill Tishey relates the dynamics of this in his *Z-Best* column. I have to admit to being an instigator, but the central figure, Gene Pizzetta, accepted a challenge to produce a new tool, accepted ideas from many people and produced a fine program. It is so good, in fact, that it is now standard issue on the Z-SUS catalog disk to provide an "on-line" search capability of the listings. As you may also know, Bill is the editor of Z-SUS, so his interest in the development process of new software comes naturally.

Al Hawley, the sysop of Z-Node Central in Los Angeles, California, debuts in this issue with a series on assembly language for the high level language programmer. This is a twist on the topic as such people are already tuned to the concept of algorithms but tend to think at a more abstract level. It seems to me that Al's series could encourage some to get off the fence and try their hands at programming at the assembly level. This issue is a great start.

I want to take a moment to mention a publication that also serves the CP/M and Z-System community. Lee Bradley publishes *Eight Bits & Change* in Newington, Connecticut. He had some very nice things to say about *TCJ* in a recent edition. To be right about it, I need to admit that I have been enjoying Lee's work since the early issues of *Pieces of Eight*, the predecessor of the current publication. *EB&C* is less technical, and less formally produced than this publication. I find it fun to sit down in the evenings and browse through an issue. You might, also. Drop Lee a note at 24 E. Cedar Street, Newington CT 06111. Subscriptions are $15 a year in the US.

David McGlone is publishing the *Z-Letter* on a regular basis now. Unfortunately, I don't have any more information on it. I am told he does an excellent job.

Well, friends, that is about all the ramblings I have for you this issue. Again, the great articles keep coming in, though there is always room for more. We should all take pride in our collective achievements as seen in these pages. I will leave you now to enjoy the journal. Before you do, however, I want to take a brief moment away from the topic of computers and talk about someone very special to me.

A Very Special Person

Several readers noticed a fish symbol in the masthead of the last issue. The fish was a secret signal between early Christians during the time of the Roman persecutions. And yes, I am a Christian. But this is not a religious journal and I had a deeper reason for placing it there. Bear with me as I want to tell you about it.

One of the members of our Quaker Meeting for Worship founded the Central New Jersey chapter of FISH some twenty years ago. FISH feeds the poor, hungry and homeless and Anita Hoynes spent her life serving the less fortunate. When I found a moment, I would lend a hand but I never seemed to have enough time to give Anita. This last Christmas season found her organization swamped with a great many more needy families due to the deepening recession and with fewer people to help. Hers is not a small project: Anita's chapter feeds thousands of families. And so it was particularly hard for me to deny her call for help. The demands of putting out my first issue of this journal were such that I just could not be there for Anita. I was bothered greatly and as a small token, I placed the fish in the masthead.

When Anita saw her first copy of *TCJ*, she was thrilled. It pleased her to see a friend doing well. I never heard a harsh word from this woman, though I felt I had let her down. But you should have seen her eyes when she found the fish! She knew immediately why it was there. For a brief moment, I thought I saw a glimmer of a tear in her eyes. My signal was received.

This story takes a tragic turn here. Last week, Anita was violently murdered by one of the very people she spent her life serving. The loss of any life diminishes us, but losing this person has been very hard on me. We pray that we may gain our share of our Creator's Wisdom. In watching her, I found myself praying for a share of hers.

Thank you for bearing with me on this. It really does mean a lot to me. And thank you, Anita, for showing the way we were meant to live our lives on this earth. Know that our love and faith is with you.●

continued from page 44

veloping products like these are not for the garage type developer. Now when it comes to problems add an extra 20 to 40 thousand bucks for test gear. LANs require LAN sniffers to see what is actually happening. We found our interface program was combining two packets together if reset during a previous operation. Without the sniffer I doubt I would have ever found the problem.

From an installation standpoint, LAN security is a major problem. When I installed a Novel system several years ago, the major problem was getting the new users to understand the importance of security. After that step it was figuring out how to set up the structure which can become a real management problem for some small organizations. The original group was 3 people with plans to add more later. My feeling after the project was too far along to change, the LAN was over kill by many times. It cost everybody lots of time, money, and added an extra layer that new or beginner users have to learn. For three people the $25 network would most likely be the best, and after they become larger and higher skilled, only then add a LAN option.

The last or latest problem is using true blue products. We have a model 70 that has been locking up for no apparent reason. We have had previous problems with the 80 and are starting to think they are related. It seems IBM has a habit of selling systems with old problems, bad ROMs, or boards that have been recalled. The latest problems are ROM BIOS related and might be affecting the 70, 80, and 90 models. They lock up after doing disk accesses and need a special work around driver. It is not so much a problem that the IBM products have bugs, all products have some amount of problems. The main difference is finding out about it, we found out by buying one. The industry has an attitude that anything IBM does is ok and problem free. The truth is far from that.

Forth Day

A few of us went to FORTH DAY 1990 in the San Francisco bay area last month. Heard some good speakers for such a low key affair. About 30 to 40 people showed up with about 10 to 12 speakers. Chuck Moore was there and talked about his latest cpu project. He keeps making them faster and smaller. Has a whole CAD system in less than 64K of memory. The real interest for us and many others was the availability of EFORTH. That is a Forth especially set up for use in embedded systems. Dr. Ting had a hand in it's production as well as selling books about it.

You can download the files from the GEnie Forth conference as well as many other places. I am in the process of porting it to the 68K system at work to check out the use of it instead of our old debuggers. It has 29 to 31 words that require redoing in assembler for whichever CPU type you are using. All the other words are high level and would not need changing. The documents with it suggest about a month to port it over, but it looks like mine will be ready to try in one or two days of work. What will take long to do is setting up the code for ROM use, where it will require closer checking of RAM and ROM usage. They have a sample 8051 version to show how simple it can be. The code runs on a PC and they use standard MASM 5.1 for both the PC and 8051 version. The idea is to use define statements for other CPUs since only 31 words maximum need coding. That coding can also be rather simple so I would agree that for some CPU type just use defines. The most interesting concept stated was that metacompling was dead. It proved far too complex for most people to learn and just turned users away. Eforth is assembler based and I am glad to see more people realizing that assembler is the best way to get systems up, and not metacompling (metacompling uses a Forth system to generate a new Forth system).

Overall our opinion of the Forth Day was rather low key. The people seemed tired, and not with over work. Many of us Forth people get tired of being put down by C people when we can prove how much faster and better the code use is when in Forth. One bright spot was the talk about how SUN Computer Systems is using Forth in every one of their systems. Pretty soon there will be more embedded Forth systems out there than any other language. The flip side of that is that most SUN user have little if any knowledge of that fact. Once the system boots and works properly they have little contact with Forth. For Sun it has been a good move that has saved money and provided lots of extra benefits. For the Forth community it will probably help in the long run, but for now just try and get a job doing Forth programming. Employers want C and more C programmers, nothing else will do (mainly because they are cheap and plentiful).

Time to Go

I guess I have said enough for now. Hopefully I have got some of you to rethink whether or not new technology is always better. I still remember attending a talk by Schumacker of "SMALL IS BEAUTIFUL" fame. Lately I have been associating some of his ideas against the direction and type of products the computer industry is turning out. I have also found out that my best running and trouble free programs are those which are small and simple (also usually in Forth!). The bigger and more complex they get it seems the worse they work. Sounds like small is beautiful after all.●

Cross-Assemblers as low as $50.00
Simulators as low as $100.00
Cross-Disassemblers as low as $100.00
Developer Packages
as low as $200.00(a $50.00 Savings)

A New Project
Our line of macro Cross-assemblers are easy to use and full featured, including conditional assembly and unlimited include files.

Get It To Market--FAST
Don't wait until the hardware is finished to debug your software. Our Simulators can test your program logic before the hardware is built.

No Source!
A minor glitch has shown up in the firmware, and you can't find the original source program. Our line of disassemblers can help you re-create the original assembly language source.

Set To Go
Buy our developer package and the next time your boss says "Get to work.", you'll be ready for anything.

Quality Solutions
PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

BROAD RANGE OF SUPPORT
- Currently we support the following microprocessor families (with more in development):

Intel 8048	RCA 1802,05	Intel 8051	Intel 8096
Motorola 6800	Motorola 6801	Motorola 68HC11	Motorola 6805
Hitachi 6301	Motorola 6809	MOS Tech 6502	WDC 65C02
Rockwell 65C02	Intel 8080,85	Zilog Z80	NSC 800
Hitachi HD64180	Motorola 68000,8	Motorola 68010	Intel 80C196

- All products require an IBM PC or compatible.

So What Are You Waiting For? Call us:
PseudoCorp
Professional Development Products Group
716 Thimble Shoals Blvd, Suite E
Newport News, VA 23606
(804) 873-1947 FAX: (804)873-2154

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Special Close Out Sale on these back issues only.

3 or more, $1.50 each postpaid in the US or $3.00 postpaid airmail outside US.

Issue Number 1:

- RS-232 Interface Part 1
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:

- File Transfer Programs for CP/M
- RS-232 Interface Part 2
- Build Hardware Print Spooler Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:

- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:

- Optronics, Part 1: Detecting, Generating and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 8:

- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board, Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 18:

- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1

Issue Number 19:

- Using the Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part 7
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking & Windows with CP/M: A Review of MTBASIC

Issue Number 20:

- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K

Issue Number 21:

- Extending Turbo Pascal: Customize with Procedures & Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC

Issue Number 22:

- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column

Issue Number 23:

- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- The Ampro Little Board Column

Issue Number 24:

- Selecting & Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assemble Code for CP/M
- The C Column: Software Text Filters
- Ampro 186 Column: Installing MS-DOS Software
- The Z-Column
- NEW-DOS: The CCP Internal Commands
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Issue Number 25:

- Repairing & Modifying Printed Circuits
- Z-Com vs. Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro LB
- Building a SCSI Adapter
- NEW-DOS: CCP Internal Commands
- Ampro 186 Networking with SuperDUO
- ZSIG Column

Issue Number 26:

- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS

Issue Number 27:

- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis & Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program in Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying the CP/M Disk Parameter Block for Foreign Disk Formats

Issue Number 28:

- Starting Your Own BBS
- Build an A/D Converter for the Ampro Little Board
- HD64180: Setting the Wait States & RAM Refresh using PRT & DMA
- Using SCSI for Real Time Control
- Open Letter to STD Bus Manufacturers
- Patching Turbo Pascal
- Choosing a Language for Machine Control

Issue Number 29:

- Better Software Filter Design
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 1
- Using the Hitachi hd64180: Embedded Processor Design
- 68000: Why use a new OS and the 68000?
- Detecting the 8087 Math Chip
- Floppy Disk Track Structure
- The ZCPR3 Corner

Issue Number 30:

- Double Density Floppy Controller
- ZCPR3 IOP for the Ampro Little Board
- 3200 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 2
- Non-Preemptive Multitasking
- Software Timers for the 68000
- Lilliput Z-Node
- The ZCPR3 Corner
- The CP/M Corner

Issue Number 31:

- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying Operating Systems
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:

- Language Development: Automatic Generation of Parsers for Interactive Systems
- Designing Operating Systems: A ROM based OS for the Z81
- Advanced CP/M: Boosting Performance
- Systematic Elimination of MS-DOS Files: Part 1, Deleting Root Directories & an In-Depth Look at the FCB
- WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII Terminal Based Systems
- K-OS ONE and the SAGE: System Layout and Hardware Configuration
- The ZCPR3 Corner: NZCOM and ZCPR34

Issue Number 33:

- Data File Conversion: Writing a Filter to Convert Foreign File Formats
- Advanced CP/M: ZCPR3PLUS & How to Write Self Relocating Code
- DataBase: The First in a Series on Data Bases and Information Processing
- SCSI for the S-100 Bus: Another Example of SCSI's Versatility
- A Mouse on any Hardware: Implementing the Mouse on a Z80 System
- Systematic Elimination of MS-DOS Files: Part 2, Subdirectories & Extended DOS Services
- ZCPR3 Corner: ARUNZ Shells & Patching WordStar 4.0

Issue Number 34:

- Developing a File Encryption System.
- Database: A continuation of the data base primer series.
- A Simple Multitasking Executive: Designing an embedded controller multitasking executive.
- ZCPR3: Relocatable code, PRL files, ZCPR34, and Type 4 programs.
- New Microcontrollers Have Smarts: Chips with BASIC or Forth in ROM are easy to program.
- Advanced CP/M: Operating system extensions to BDOS and BIOS, RSXs for CP/M 2.2.
- Macintosh Data File Conversion in Turbo Pascal.
- The Computer Corner

Issue Number 35:

- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assem. source code.
- Real Computing: The NS32032.
- S-100: EPROM Burner project for S-100 hardware hackers.
- Advanced CP/M: An up-to-date DOS, plus details on file structure and formats.
- REL-Style Assembly Language for CP/M and Z-System. Part 1: Selecting your assembler, linker and debugger.
- The Computer Corner

Issue Number 36:

- Information Engineering: Introduction.
- Modula-2: A list of reference books.
- Temperature Measurement & Control: Agricultural computer application.
- ZCPR3 Corner: Z-Nodes, Z-Plan, Amstrand computer, and ZFILE.
- Real Computing: NS32032 hardware for experimenter, CPUs in series, software options.
- SPRINT: A review.
- REL-Style Assembly Language for CP/M & ZSystems, part 2.
- Advanced CP/M: Environmental programming.
- The Computer Corner.

Issue Number 37:

- C Pointers, Arrays & Structures Made Easier: Part 1, Pointers.
- ZCPR3 Corner: Z-Nodes, patching for NZCOM, ZFILER.
- Information Engineering: Basic Concepts: fields, field definition, client worksheets.
- Shells: Using ZCPR3 named shell variables to store date variables.
- Resident Programs: A detailed look at TSRs & how they can lead to chaos.
- Advanced CP/M: Raw and cooked console I/O.
- Real Computing: The NS 32000.
- ZSDOS: Anatomy of an Operating System: Part 1.
- The Computer Corner.

Issue Number 38:

- C Math: Handling Dollars and Cents With C.
- Advanced CP/M: Batch Processing and a New ZEX.
- C Pointers, Arrays & Structures Made Easier: Part 2, Arrays.
- The Z-System Corner: Shells and ZEX, new Z-Node Central, system security under Z-Systems.
- Information Engineering: The portable Information Age.
- Computer Aided Publishing: Introduction to publishing and Desk Top Publishing.
- Shells: ZEX and hard disk backups.
- Real Computing: The National Semiconductor NS320XX.
- ZSDOS: Anatomy of an Operating System, Part 2.

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Issue Number 39:

· Programming for Performance: Assembly Language techniques.
· Computer Aided Publishing: The Hewlett Packard LaserJet.
· The Z-System Corner: System enhancements with NZCOM.
· Generating LaserJet Fonts: A review of Digi-Fonts.
· Advanced CP/M: Making old programs Z-System aware.
· C Pointers, Arrays & Structures Made Easier: Part 3: Structures.
· Shells: Using ARUNZ alias with ZCAL.
· Real Computing: The National Semiconductor NS320XX.
· The Computer Corner.

Issue Number 40:

· Programming the LaserJet: Using the escape codes.
· Beginning Forth Column: Introduction.
· Advanced Forth Column: Variant Records and Modules.
· LINKPRL: Generating the bit maps for PRL files from a REL file.
· WordTech's dBXL: Writing your own custom designed business program.
· Advanced CP/M: ZEX 5.0•The machine and the language.
· Programming for Performance: Assembly language techniques.
· Programming Input/Output With C: Keyboard and screen functions.
· The Z-System Corner: Remote access systems and BDS C.
· Real Computing: The NS320XX
· The Computer Corner.

Issue Number 41:

· Forth Column: ADTs, Object Oriented Concepts.
· Improving the Ampro LB: Overcoming the 88Mb hard drive limit.
· How to add Data Structures in Forth
· Advanced CP/M: CP/M is hacker's haven, and Z-System Command Scheduler.
· The Z-System Corner: Extended Multiple Command Line, and aliases.
· Programming disk and printer functions with C.
· LINKPRL: Making RSXes easy.
· SCOPY: Copying a series of unrelated files.
· The Computer Corner.

Issue Number 42:

· Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
· Using BYE with NZCOM.
· C and the MS-DOS Screen Character Attributes.
· Forth Column: Lists and object oriented Forth.
· The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
· 68705 Embedded Controller Application: An example of a single-chip microcontroller application.
· Advanced CP/M: PluPerfect Writer and using BDS C with REL files.
· Real Computing: The NS 32000.
· The Computer Corner

Issue Number 43:

· Standardize Your Floppy Disk Drives.
· A New History Shell for ZSystem.
· Heath's HDOS, Then and Now.
· The ZSystem Corner: Software update service, and customizing NZCOM.
· Graphics Programming With C: Graphics routines for the IBM PC, and the Turbo C graphics library.
· Lazy Evaluation: End the evaluation as soon as the result is known.
· S-100: There's still life in the old bus.
· Advanced CP/M: Passing parameters, and complex error recovery.
· Real Computing: The NS32000.
· The Computer Corner.

Issue Number 44:

· Animation with Turbo C Part 1: The Basic Tools.
· Multitasking in Forth: New Micros F68FC11 and Max Forth.
· Mysteries of PC Floppy Disks Revealed: FM, MFM, and the twisted cable.
· DosDisk: MS-DOS disk format emulator for CP/M.
· Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
· Real Computing: The NS32000.
· Forth Column: Handling Strings.
· Z-System Corner: MEX and telecommunications.
• The Computer Corner

Issue Number 45:

· Embedded Systems for the Tenderfoot: Getting started with the 8031.
· The Z-System Corner: Using scripts with MEX.
· The Z-System and Turbo Pascal: Patching TURBO.COM to access the Z-System.
· Embedded Applications: Designing a Z80 RS-232 communications gateway, part 1.
· Advanced CP/M: String searches and tuning Jetfind.
· Animation with Turbo C: Part 2, screen interactions.
· Real Computing: The NS32000.
· The Computer Corner.

Issue Number 46:

· Build a Long Distance Printer Driver.
· Using the 8031's built-in UART for serial communications.
· Foundational Modules in Modula 2.
· The Z-System Corner: Patching The Word Plus spell checker, and the ZMATE macro text editor.
· Animation with Turbo C: Text in the graphics mode.
· Z80 Communications Gateway: Prototyping, Counter/Timers, and using the Z80 CTC.

Issue Number 47:

· Controlling Stepper Motors with the 68HC11F
· Z-System Corner: ZMATE Macro Language
· Using 8031 Interrupts
· T-1: What it is & Why You Need to Know
· ZCPR3 & Modula, Too
· Tips on Using LCDs: Interfacing to the 68HC705
· Real Computing: Debugging, NS32 Multitasking & Distributed Systems
· Long Distance Printer Driver: correction
· ROBO-SOG 90
· The Computer Corner

Issue Number 48:

· Fast Math Using Logarithms
· Forth and Forth Assembler
· Modula-2 and the TCAP
· Adding a Bernoulli Drive to a CP/M Computer (Building a SCSI Interface)
· Review of BDS "Z"
· PMATE/ZMATE Macros
· Real Computing
· Z-System Corner: Patching MEX-Plus and TheWord, Using ZEX
· Z-Best Software
· The Computer Corner

	U.S.	Foreign (Surface)	Foreign (Airmail)	Total
Subscriptions				
1year (6 issues)	$18.00	$24.00	$38.00	_____
2 years (12 issues)	$32.00	$44.00	$72.00	_____
Back Issues				
18 thru #43	$3.50 ea.		$5.00 ea.	_____
6 or more	$3.00 ea.		$4.50 ea.	_____
#44 and up	$4.50 ea.		$6.00 ea.	_____
6 or more	$4.00 ea.		$5.50 ea.	_____

Issue #s ordered_____

Subscription Total _____

Back Issues Total _____

Total Enclosed _____

Name_____

Address_____

_____

_____

Payment is accepted by check or money order. Checks must be in US funds, drawn on a US bank. Personal checks within the US are welcome.

The Computer Journal
P.O. Box 12, S. Plainfield, NJ 07080-0012
Phone (908) 755-6186

The Computer Corner

By Bill Kibler

Well it is move time for me. Just moved to a larger place, but my computer space is actually smaller. Looks like it will take some time to get set up again, but we are now on a dead end road with considerable peace and quite as well as room to spread out later.

Went to a computer swap last week and sold mostly disk drives and cabinets. Seems most shoppers are not interested in parts of systems any more, just PC based plug-in units. That also agrees with my part time teaching in electronics. The college is planning on changing the slant of the program to a more black box approach.

Enrollment in pure electronic courses has fallen rather sharply, so the department is changing from an electronic tech approach to a computer or industrial computer maintenance approach. The students will spend their time doing mostly black box style repairs. No longer is it necessary to be able to trouble shoot to the component level. Most systems these days can not be repaired to the component level so teaching that operation is proving pointless.

No longer is it necessary to be able to trouble shoot to the component level. Most systems these days can not be repaired to the component level so teaching that operation is proving pointless.

This point was brought home when I went to start the last semesters teaching. My classroom had been taken over by the computer department. I found myself stuck in a regular classroom and the two electronic labs were now only one. The electronics department has two instructors, while the computer group has gone from 4 to 6 and may add one more next fall. Computer science is not slowing down around here.

Hardware Talk

While at a recent family get together, I found that computers can creep into the conversation as well. I was asked several times about what current level or model one should get. Personally, I still am not too happy with the PC clones but they are the cheapest still. I prefer the 68000 based units for their better software, but alas the PC market has more overall appeal for the beginner and college bound student. One of my brothers has two college bound students and they are finding out how many of the instructors are using and requiring the students to use PC based programs.

The cost and availability of PC clones is so great that despite their draw backs they are by far the best buy. One of the people at work attended the computer show in Las Vegas, where he saw several sharp items. We still think the new Atari machines are the thing to have, as they can run Mac and DOS as well as the Atari ST software. The way they do this is using co-processor boards, the PC being a 386 plug in. The Macintosh is using the real Macintosh ROMs. So the ST becomes simply I/O for the other processes.

For me personally, I have been thinking more and more along the lines of multiple processor systems. I do not have the room to have several systems for each of the many different areas I may have to deal with. My wife is a teacher and wants to run Apple II programs and Macintosh graphic stuff (she is an art teacher as well as a graphic artist). My teaching is both Macintosh and PC clone. I work on clones running 68K co-processors talking to Tandem mainframes. My real interest is in embedded systems which could be anything from 6805 to RTX2000s. I currently have hardware to support all but the Apple and Macs, and I plan to get a adapter card for my Atari ST to cover them. As you can guess, that is a lot of physical systems, especially when you throw in the older CP/M S-100 boxes. Takes up lots of room.

What I am thinking about is retooling the old S-100 products (or some other bus) to be able to put together one system that would run all the processors I am working with. I can currently run the Xerox CP/M programs on a PC clone and some special co-processors. There are several programs and adapters for the Atari ST to run Macintosh, CP/M, 8 bit Atari, PC/DOS, and many more. Apple is getting into the picture with a cheaper machine to run Apple II and Macintosh programs. The way things are going, I will be able to buy what I want before I could adapt and make such a system for myself. I sure hope so!

Problems

Work has been dragging along these days. Lots of little problems keep appearing to slow things down. We run co-processors in IBM PC based machines. Our latest project is changing our serial data path to LAN based usage. We ended up using a LAN sniffer the other day for one problem. The type and nature of test equipment is getting more and more complex everyday. I can't stress the importance of not going to higher tech solutions if there is some other way around the problem. To me, LANs are a good example of why I believe in that statement.

First lets talk cost of development. We found that testing our program required at least 32 systems be on line before certain types of problems occurred. The cost of each system, even at OEM prices is staggering. The PC's are PS2 386's, plus LAN cards, our co-processor board, and added memory. Our cost is over $7,000 each, and to do tests properly you need over 32 units or over $200,000 of capital outlay. De

continued page 41

Plu*Perfect Systems == World-Class Software

BackGrounder ii **$75**

Task-switching ZCPR34. Run 2 programs, cut/paste screen data. Use calculator, notepad, screendump, directory in background. CP/M 2.2 only. Upgrade licensed version for $20.

Z-System **$69.95**

The renowned Z-System command processor (ZCPR v 3.4) and companion utilities. Dynamically change memory use. Installs automatically
Order **Z3PLUS** for CP/M Plus, or **NZ-COM** for CP/M 2.2.

ZMATE **$50**

New Z-System version of renowned PMATE macro editor with split-screen mode for two-window viewing of one or more files. Extremely powerful and versatile macro capability lets you automate repetitive or complex editing tasks, making it the ultimate programmer's editor. Macros can be saved for reuse and also assigned to keys. Editing keys can be reconfigured for personal style. Supports drive/user and named-directory file references. Auto-installs on Z systems. Z-80 only. Supplied with user manual and sample macro files.

PluPerfect Writer **$35**

Powerful text and program editor with EMACS-style features. Edit files up to 200K. Use up to 8 files at one time, with split-screen view. Short, text-oriented commands for fast touch-typing: move and delete by character, word, sentence, paragraph, plus rapid insert/delete/copy and search. Built-in file directory, disk change, space on disk. New release of our original upgrade to Perfect Writer 1.20, now for all Z80 computers. On-disk documentation only.

ZSDOS ..... **$75, for ZRDOS users just $60**

State-of-the-art operating system. Built-in file DateStamping. Fast hard-disk warmboots. Menu-guided installation. Enhanced time and date utilities. CP/M 2.2 only.

DosDisk ..... **$30 - $45**

Use MS-DOS disks without copying files. Subdirectories too. Kaypro w/TurboRom, Kaypro w/KayPLUS, MD3, MD11, Xerox 820-I w/Plus 2, ON!, C128 w/1571 -- $30. SB180 w/XBIOS -- $35. Kit -- $45. Kit requires assembly language expertise and BIOS source code.

MULTICPY **$45**

Fast format and copy 90+ 5.25" disk formats. Use disks in foreign formats. Includes DosDisk. Requires Kaypro w/TurboRom.

JetFind **$50**

Fastest possible text search, even in LBR, squeezed, crunched files. Also output to file or printer. Regular expressions.

To order: Specify product, operating system, computer, 5 1/4" disk format. Enclose **check**, adding $3 shipping ($5 foreign) + 6.5% tax in CA. Enclose invoice if upgrading BGii or ZRDOS.

Plu*Perfect Systems
410 23rd St.
Santa Monica, CA 90402
(213)-393-6105 (eves.)

BackGrounder II ©, DosDisk ©, Z3PLUS ©, PluPerfect Writer ©, JetFind © Copyright 1986-88 by Bridger Mitchell.

SAGE MICROSYSTEMS EAST

Selling & Supporting the Best in 8-Bit Software

- Automatic, Dynamic, Universal Z-Systems
  - Z3PLUS: Z-System for CP/M-Plus computers ($70)
  - NZCOM: Z-System for CP/M-2.2 computers ($70)
  - ZCPR34 Source Code: if you need to customize ($50)
- ZSUS: Z-System Software Update Service, public-domain software distribution service (write for a flyer with full information)
- Plu*Perfect Systems
  - Backgrounder ii: CP/M-2.2 multitasker ($75)
  - ZSDOS/ZDDOS: date-stamping DOS ($75, $60 for ZRDOS owners)
  - ZSDOS Programmer's Manual ($10)
  - DosDisk: MS-DOS disk-format emulator, supports subdirectories and date stamps ($30 standard, $35 XBIOS BSX, $45 kit)
  - JetFind: super fast, extemely flexible text file scanner ($50)
- ZMATE: macro text editor / customizable wordprocessor ($50)
- PCED — the closest thing to ARUNZ and LSH (and more) for MS-DOS ($50)
- BDS C — including special Z-System version ($90)
- Turbo Pascal — with new loose-leaf manual ($60)
- SLR Systems (The Ultimate Assembly Language Tools)
  - Z80 assemblers using Zilog (Z80ASM), Hitachi (SLR180), or Intel (SLRMAC) mnemonics
  - linker: SLRNK
  - TPA-based ($50 each) or virtual-memory (special: $160 each)
- ZMAC — Al Hawley's Z-System macro assembler with linker and librarian ($50 disk, $70 with printed manual)
- NightOwl (advanced telecommunications, CP/M and MS-DOS versions)
  - MEX-Plus: automated modem operation with scripts ($60)
  - MEX-Pack: remote operation, terminal emulation ($100)

Next-day shipping of most products with modem download and support available. Order by phone, mail, or modem. Shipping and handling $3 per order (USA). Check, VISA, or MasterCard. Specify exact disk format.

Sage Microsystems East
1435 Centre St., Newton Centre, MA 02159-2469
Voice: 617-965-3552 (9:00am - 11:30pm)
Modem: 617-965-7259 (pw=DDT) (MABOS on PC-Pursuit)